

On the van der Waerden numbers $w(2; 3, t)$

Tanbir Ahmed^a, Oliver Kullmann^b, Hunter Snevily^c

^aDepartment of Computer Science and Software Engineering, Concordia University, Montréal, Canada.
ta_ahmed@cs.concordia.ca

^bComputer Science Department, Swansea University, Swansea, UK.
O.Kullmann@Swansea.ac.uk

^cDepartment of Mathematics, University of Idaho - Moscow, Idaho, USA.
snevily@uidaho.edu

Abstract

In this paper we present results and conjectures on the van der Waerden numbers $w(2; 3, t)$. We have computed the exact value of the previously unknown van der Waerden number $w(2; 3, 19) = 349$, and we provide new lower bounds for $t = 20, 21, \dots, 39$, where for $t = 20, \dots, 30$, we conjecture these bounds to be exact. The lower bounds for $w(2; 3, t)$ with $t = 24, \dots, 30$ refute the conjecture that $w(2; 3, t) \leq t^2$ as suggested in [8]. Based on the known values of $w(2; 3, t)$, we investigate regularities to better understand the lower bounds of $w(2; 3, t)$. We also provide heuristics to generate examples of good partitions for $w(2; 3, t)$. Motivated by such heuristics, we introduce *palindromic van der Waerden numbers* $\text{pd}w(k; t_0, \dots, t_{k-1})$, which are defined as the ordinary numbers $w(k; t_0, \dots, t_{k-1})$, but where only palindromic solutions are considered, reading the same from both ends. Since the underlying property is non-monotonic, these numbers are actually pairs of numbers. We compute $\text{pd}w(2; 3, t)$ for $3 \leq t \leq 25$, and we provide bounds for $26 \leq t \leq 39$, which we believe to be exact for $t \leq 35$. All computations are based on SAT solving, and we discuss the various relations between SAT solving and Ramsey theory.

1. Introduction

We consider Ramsey theory and its connections to computer science (see [45] for a survey) by exploring a rather recent link, especially to algorithms and formal methods, namely to “SAT solving”. SAT is the problem of finding a satisfying assignment for a propositional formula. Since Ramsey problems can naturally be formulated as SAT problems, SAT solvers can be used to compute numbers from Ramsey theory. In the present article, we consider van der Waerden numbers, where SAT had its biggest success in Ramsey theory: determination of $w(2; 6, 6)$ to be 1132 [31], a new diagonal van der Waerden number after almost 30 years.

The *van der Waerden number* $w(k; t_0, t_1, \dots, t_{k-1})$ is the smallest n such that for any partition $\{1, 2, \dots, n\} = P_0 \cup P_1 \cup \dots \cup P_{k-1}$ there exists a j in $\{0, 1, \dots, k-1\}$ such that P_j contains an arithmetic progression of length t_j . A *good partition* of the set $\{1, 2, \dots, n\}$ corresponding to $w(k; t_0, t_1, \dots, t_{k-1})$ contains no block P_j with an arithmetic progression of length t_j (for any j). In this paper, we are interested in the specific van der Waerden numbers $w(2; 3, t)$, $t \geq 3$. The known exact values of $w(2; 3, t)$ with $t = 3, 4, \dots, 18$ are

$$9, 18, 22, 32, 46, 58, 77, 97, 114, 135, 160, 186, 218, 238, 279, \text{ and } 312.$$

As references and for relevant information on the above numbers, see Chvátal [9], Brown [7], Beeler and O’Neil [4], Kouril [31], Landman, Robertson and Culver [40], and Ahmed [2, 3]. Recently, Kullmann [35]¹ reported the following lower bounds

$$w(2; 3, 19) \geq 349, w(2; 3, 20) \geq 389, w(2; 3, 21) \geq 416.$$

We confirm the exact value of $w(2; 3, 19) = 349$, and we extend the list of lower bounds up to $t = 39$. Brown, Landman, and Robertson [8], provided the theoretical lower bound $w(2; 3, t) \geq t^{2-o(1)}$, and observed that $w(2; 3, t) \leq t^2$ for $5 \leq t \leq 16$. Our experiments show that this assumption is not true for all t . We provide an improved upper bound as a conjecture (satisfying all known values and lower bounds of $w(2; 3, t)$).

¹the conference article [36] contains only material related to Green-Tao numbers and SAT

1.1. Using SAT solvers

As explored in Dransfield et al. [13], Herwig et al. [18], Kouril [31], Ahmed [2, 3], and Kullmann [35, 36], we can generate an instance F of the satisfiability problem (for definition, see any of the above references) corresponding to $w(k; t_0, t_1, \dots, t_{k-1})$ and integer n , such that F is satisfiable if and only if $n < w(k; t_0, t_1, \dots, t_{k-1})$. In particular, an instance corresponding to $w(2; 3, t)$ with n variables can be generated with the following clauses:

- (a) $\{x_a, x_{a+d}, x_{a+2d}\}$ with $a \geq 1, d \geq 1, a + 2d \leq n$, and
- (b) $\{\bar{x}_a, \bar{x}_{a+d}, \dots, \bar{x}_{a+d(t-1)}\}$ with $a \geq 1, d \geq 1, a + d(t-1) \leq n$,

where $x_i = \varepsilon$ encodes $i \in P_\varepsilon$ for $\varepsilon \in \{0, 1\}$ (if x_i is not assigned but the formula is satisfied, then i can be arbitrarily placed in either of the blocks of the partition). Clauses (a) prohibit the existence of an arithmetic progression of length 3 in P_0 and clauses (b) prohibit the existence of an arithmetic progression of length t in P_1 .

To check the satisfiability of the generated instance, we need to use an algorithm that either solves the instance and provides a satisfying assignment, or says that the formula is unsatisfiable. We have at our disposal two kinds of algorithms: complete and incomplete. A complete algorithm like DPLL (see [12, 11]; Algorithm 1 is a slightly modified variant given in [3]) finds a satisfying assignment if one exists, and otherwise correctly says that no satisfying assignment exists and the formula is unsatisfiable. SAT solving has progressed much beyond this simple algorithm (see the handbook [6] for general information), however on this special problem class this basic algorithm together with a basic heuristics (see Subsection 2.1) is very competitive; see Section 5 for more information on SAT solvers in this context (and see [50] for applications of SAT to combinatorics).

Algorithm 1 DPLL algorithm

```

1: function DPLL( $F$ )
2:   while TRUE do
3:     if  $\{u\} \in F$  and  $\{\bar{u}\} \in F$  then return UNSATISFIABLE
4:     else if there is a clause  $\{v\}$  then  $F = F|v$ 
5:     else break
6:   end while
7:   if  $F = \emptyset$  then return SATISFIABLE
8:   Choose an unassigned literal  $u$  using a branching rule
9:   if DPLL( $F|u$ ) = SATISFIABLE then return SATISFIABLE
10:  if DPLL( $F|\bar{u}$ ) = SATISFIABLE then return SATISFIABLE
11:  return UNSATISFIABLE
12: end function

```

Given a formula F and a literal u in F , we let $F|u$ denote the *residual formula* arising from F when u is set to true: explicitly, this formula is obtained from F by (i) removing all the clauses that contain u , (ii) deleting \bar{u} from all the clauses that contain \bar{u} , (iii) removing both u and \bar{u} from the list of literals. Each recursive call of DPLL may involve a choice of a literal u . Algorithms for making these choices are referred to as *branching rules*. It is customary to represent each call of DPLL(F) by a node of a binary tree. By branching on a literal u , we mean calling DPLL($F|u$). If this call leads to a contradiction, then we call DPLL($F|\bar{u}$). Every node that is not a leaf has at least one child and may not have both children. We refer to this tree as *DPLL-tree* of F .

Local-search based incomplete algorithms (see Ubsat-suite [49]) are generally faster (as they try to find a satisfying assignment as fast as possible when we can control the heuristic, number of iterations, runs, and several other parameters) than a DPLL-like algorithm, but may fail to deliver a satisfying assignment when there exists one. A good partition is a proof of a lower bound for a certain van der Waerden number irrespective of its means of achievement. So incomplete algorithms are handy for obtaining good partitions and improving lower bounds of van der Waerden numbers. When they fail to improve the lower bound any further, we need to turn to a complete algorithm.

1.1.1. Parallel/distributed SAT solving

The problems we consider are computationally hard, and for the hardest of them in this paper, computation of $w(2; 3, 19) = 349$, a single processor, even when run for a long time, is by far not enough. Hence some form of

parallelisation or distribution of the work is needed. Four levels of parallelisation have been considered for SAT solving (in a variety of schemes):

- (i) Processor-level parallelisation: This helps only for very special algorithms, and can only achieve some relatively small speed-up; see [23] for an example which exploits parallel bit-operations. It seems to play no role for the problems we are considering.
- (ii) Computer-level parallelisation: Here it is exploited that currently a single (standard) computer can contain up to, say, 16 relatively independent processing units, working on shared memory. So threads (or processes) can run in parallel, using one (or more) of the following general forms of collaboration:
 - (a) Partitioning the work via partitioning the instance (see below); [51, 28] are “classical” examples.
 - (b) Using the same algorithm running in various nodes on the same problem, exploiting randomisation and/or sharing of learned results; see [25, 17] for recent examples.
 - (c) Using some portfolio approach, running different algorithms on the same problem, exploiting that various algorithms can behave very differently and unpredictably; see [16] for the first example.

Often these approaches are combined in various ways; see [47, 15, 26, 27] for recent examples. Approaches (b) and (c) do not seem to be of much use for the well-specified problem domain of hard instances from Ramsey theory. Only (a) is relevant, but in a more extreme form (see below). In the context of (ii), still only relatively “easy” problems (compared to the hard problems from Ramsey theory) are tackled.

- (iii) Parallelisation on a cluster of computers: Here up to, say, 100 computers are considered, with restricted communication (though typically still non-trivial). In this case, the approach (ii)(a) becomes more dominant, but other considerations of (ii) are still relevant. For hard problems this form of computation is a common approach.
- (iv) Internet computation, with completely independent computers, and only very basic communication between the centre and the machines: Here only (ii)(a) is applicable, and, in principle, the number of computers is unbounded; yet there is no real example for a SAT computation at this level.

We remark that the classical area of “high performance computing” seems to be of no relevance for SAT solving, since the basic SAT algorithms like unit-clause propagation are, different from typical forms of numerical computation, inherently sequential (compare also our remarks to (i)).

Now a major advantage of the DPLL solver architecture discussed in Subsection 1.1 (which has been further developed into so-called “look-ahead” SAT solvers) is that the computation is easily parallelisable and distributable: Just compute the tree only up to a certain depth d , and solve the (up to) 2^d sub-problems at level d . Only minimal interaction is required: The sub-problems are solved independently, and in case one sub-problem has been found satisfiable, then the whole search can be aborted (for the purpose of mere SAT-solving; for counting all solutions of course the search needs to be completed). And the sub-problems are accessible via the partial assignment constituting the path from the root to the corresponding leaf, and thus also require only small storage space. This is the core of method (ii)(a) from above, and will be further considered in Subsection 2.1 (for our special example class).

In the subsequent subsection we will discuss the general merits of applying SAT solving to (hard) Ramsey problems. One spin-off of this combination lies in pushing the frontier of large computations. As a first example we have developed in [22], motivated by the considerations of the present article, an improved method for (ii)(a), which is also relevant for industrial problems (typically from the verification area). One aspect exploited here is that for extremely hard problems splitting into millions of sub-instances is needed. In the literature until now (see above for examples) only splitting as required, by at most hundreds of processors, has been performed, while it turned out that the above “extreme splitting”, when combined with “modern” SAT solvers, is also beneficial in less extreme settings, and on a wide range of examples. See [37] for the elaboration of this method in the context of the problem classes of the present article.

1.1.2. Synergies between Ramsey theory and SAT

For Ramsey-numbers (see [44] for an overview on exact results), relatively precise asymptotic bounds exist, and due to the inherent symmetry, relatively specialised methods for solving concrete instances have an advantage. Vander-Waerden-like numbers seem harder to tackle, both asymptotically and exactly, and perhaps the only way ever to know the precise values is by computation (and perhaps this is also true for Ramsey-numbers, only more structures are to be exploited). SAT solvers are especially suited for the task, since the computational problems are hypergraph-colouring problems, which, at least for two colours, have canonical translations into SAT problems (as only considered

in this paper). For more colours, see the approach started in [36], while for a general theory of multi-valued SAT close to hypergraph-colouring, see [38, 39].

Through SAT computations (as in the present article), Ramsey theory acquires an applied side. Whenever for example a recent microchip is employed, this likely involves SAT solving, playing an important (though typically hidden) role in its development by yielding the underlying “engines” for its verification. See the recent handbook [6] to get some impression of this astounding development. We believe that problem instances from Ramsey theory are good benchmarks, serving to improve SAT solvers on hard instances:

- Unlike with random instances (see [1] for an overview), instances from Ramsey theory are “structured” in various ways.
- In this paper, we consider two instance classes: instances related to ordinary van der Waerden numbers $w(2; 3, t)$ and instances related to the palindromic forms $\text{pd}w(2; 3, t)$. Now already with these two classes, the two main types of complete SAT solvers, “look-ahead” (see [21]) and “conflict-driven” (see [42]), are covered in the sense that they dominate on one class each (and are (relatively) efficient); see Section 5 for further details. On the other hand, for random instances only look-ahead solvers are efficient (for complete solvers).
- Like with random instances we get scalability, though due to the possibly large and unknown growth of Ramsey-like numbers, the control of the parameters is more complicated here. Thus it is only through computational studies like this paper, serving to calibrate the scale via precise numerical data, that the field of SAT instances from Ramsey-theory becomes accessible.
- Especially for local-search methods (see [29] for an overview), these problems are hard, but not overwhelmingly so (for the ranges considered), and thus all the given upper bounds can trigger further progress (and insight) into the solution process in a relatively simple engineering-like manner.
- On the other hand, for lower bounds we need to show unsatisfiability, which is much harder. All applications of SAT solving in hardware verification are “unsatisfiability-driven” (see [5, 32] for introductions). So future progress in solving hard Ramsey instances might have the potential to trigger a breakthrough in tackling unsatisfiability, and should then also improve these industrial applications.

We believe that for better SAT solving, established hard problem instances are needed in a great variety, and we believe that Ramsey theory offers this potential. To begin the process of applying Ramsey theory in this direction, problem instances from this paper (as well as related to [36]) have been used in the SAT 2011 competition (<http://www.satcompetition.org/2011/>). As already mentioned in the previous subsection, the first fruits of the collaboration between SAT and Ramsey theory appeared in [22, 37], yielding a method for tackling hard problems with unprecedented scalability.

Finally the interaction between Ramsey theory and SAT should yield new insights for Ramsey theory itself:

1. The numerical data can yield conjectures on growth rates; see Subsection 2.4.
2. The good partitions found can yield conjectures on patterns; see Section 3.
3. New forms of Ramsey problems can be found through algorithmic considerations; see Section 4.
4. The SAT solving process, considered *in detail*, acts like a microscope, enabling insights into the structure of the problem instances which are out of sight for Ramsey theory yet. For approaches towards structures in SAT instances which we hope to study in the future see [46, 30].

1.2. The results of this paper

Section 2 contains our results on the numbers $w(2; 3, t)$. We discuss the computation of the one new van der Waerden number, and present further conjectures regarding precise values² and the growth rate. In Section 3, we investigate some patterns we found in the good partitions (establishing the lower bounds).

²to establish these conjectures will require major advances in SAT solving

In Section 4, we present a new type of van-der-Waerden-like numbers, namely *palindromic numbers*, obtained by the constraint on good partitions that they must be symmetric under reflection at the mid-point of the interval $\{1, \dots, n\}$. Perceived originally only as a heuristic tool for studying ordinary van der Waerden numbers, it turned out that these numbers are interesting objects on their own. An interesting phenomenon is that we no longer have the standard behaviour of the SAT instances with increasing n , where

- first all instances are satisfiable, and from a certain point on (the van der Waerden number) all instances are unsatisfiable,
- but now first again all instances are satisfiable, then we have a region with strict alternation between unsatisfiability and satisfiability, and only from a second point on all instances are unsatisfiable.

These two turning points constitute the palindromic number $\text{pdw}(2; 3, t)$ as pairs of natural numbers. We were able to compute $\text{pdw}(2; 3, t)$ for $t \leq 25$, exploiting now a different type of SAT solver, sometimes euphemistically called “modern SAT solvers” due to their dominance for industrial applications, the many ways to optimise them, and their erratic way of “brute-force intelligent learning”. We also provide (conjectured) values for $t \leq 39$.

Finally in Section 5, we discuss the observations on the use of the various SAT solvers involved. In Appendix A, we show how to access all the numbers and many methods in the `OKlibrary`, an open-source research platform for hard problems (around the SAT problem).

In this paper, we represent partitions of $w(2; 3, t)$ as bitstrings. For example, the partition $P_0 = \{1, 4, 5, 8\}$ and $P_1 = \{2, 3, 6, 7\}$, which is an example of a good partition of $\{1, 2, \dots, 8\}$, where $8 = w(2; 3, 3) - 1$, is represented as 01100110, or more compactly as $01^20^21^20$, using exponentiation to denote repetition of bits.

2. Computational results on $w(2; 3, t)$

This section is concerned with the numbers $w(2; 3, t)$. The discussion of the computation of $w(2; 3, 19)$ is the subject of Subsection 2.1. Conjectures on the values of $w(2; 3, t)$ for $20 \leq t \leq 30$ are presented in Subsection 2.2, while further lower bounds for $31 \leq t \leq 39$ are given in Subsection 2.3. Finally in Subsection 2.4 we update the conjecture on the (quadratic) growth of $w(2; 3, t)$.

2.1. $w(2; 3, 19) = 349$

The lower bound $w(2; 3, 19) \geq 349$ was obtained by Kullmann [35] using local search algorithms and it could not be improved any further using these incomplete algorithms (because, as we now know, the bound is tight). An example of a good partition of the set $\{1, 2, \dots, 348\}$ is as follows:

$$\begin{aligned} &1^401^601^{18}01^301^401^501^401^{11}01^901^301^601^701^501^{14}01^{16}0101^20^21^201^{15}01^401^{12}0 \\ &1^{15}01^201^501^701^{10}01^{13}01^201^{15}01^{12}01^401^{15}01^20^21^20101^901^601^{14}01^501^{14}01^2. \end{aligned}$$

To finish the search, i.e., to decide that a current lower bound of a certain van der Waerden number is exact, might require many years of CPU-time (perhaps trillions of years for the bigger lower bounds). Discovering a new van der Waerden number has always been a challenge as it requires to explore the search space completely, which has a size exponential in the number of variables in the corresponding satisfiability instance. To prove that an instance with n variables is unsatisfiable, the DPLL algorithm has to implicitly enumerate all the 2^n solutions. So the algorithm systematically explores all possible solutions, however without actually explicitly evaluating all of them — herein lies the strength (and the challenge) for SAT solving.

Dividing the computation of a search into parts that have no inter-process communication among themselves is straightforward. DPLL has this desirable property like many backtrack algorithms. We may pick a level, say ℓ , of the DPLL-tree and distribute the subtrees rooted at that level among the processors. The appropriate value of ℓ may depend on many factors like the number of computers available, and should be decided on a case-by-case basis. Distributing the tasks evenly will not guarantee the reduction of computation time by a factor close to the number of CPUs, as some of the subtrees may be considerably larger than others, or that a computer involved may be relatively slow. Advanced splitting techniques for problem-specific DPLL-tree would help to predict which branch would require to be splitted recursively. In Subsection 1.1.1 we gave an overview on this area from a general SAT perspective, and we

are concerned here with method (ii)(a). We discuss now the special structures found for the concrete instance under consideration.

The size of the DPLL-tree greatly varies with the choice of the branching rule (or heuristic). It is hard to find a branching rule that works good on every SAT instance. On a formula F , define $w(u) = \sum_k 2^{-k} d_k(u)$, where $d_k(u)$ denotes the number of clauses of length k containing u . First we find a variable x that maximises $w(x) + w(\bar{x})$, and then we choose x if $w(x) \geq w(\bar{x})$, and \bar{x} otherwise. This branching rule is known as Two-sided Jeroslaw-Wang (2sJW), (by Hooker and Vinay [24]) in the literature. Our DPLL implementation reads branching suggestions, if there is any, up to a prescribed level, and then explores the search space corresponding to the residual formula after those branches. It also periodically stores the current state of the search as a sequence of pairs (u, t) where $u \in \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$ and $t \in \{0, 1\}$. A pair $(u, 0)$ indicates that \bar{u} has to be explored, and $(u, 1)$ indicates that \bar{u} has been explored already. A generator version of the solver generates the branching suggestions up to prescribed level, and then the solver runs on different suggestions corresponding to the respective branches of the DPLL-tree. This involves a slight modification in Algorithm 1, which can easily be done. By choice of this branching rule on the formula corresponding to $w(2; 3, 19)$ with $n = 349$, for a certain value of ℓ , the parts (sub-trees of the DPLL-tree) $P_0, P_1, P_2, P_4, P_8, P_{16}, \dots$ are bigger than the others parts, and P_0 is the biggest. This observation was made in Ahmed [3] and the pattern in the sizes of the branches of the DPLL-tree helps to plan recursive splitting of the DPLL-tree to control the actual run-time.

We have proved that there is no good partition of the set $\{1, 2, \dots, 349\}$ with respect to $w(2; 3, 19)$, and hence the exactness of the lower bound $w(2; 3, 19) \geq 349$ given by Kullmann [35]. We have used 2.2 GHz AMD Opteron 64-bit processors (200 of them) from the cirrus cluster at Concordia University for running the distributed branches of the DPLL-tree corresponding to $w(2; 3, 19)$ with $n = 349$. Roughly 196 years of CPU-time has been spent. We have splitted the search space into 256 independent parts P_0, P_1, \dots, P_{255} with $\ell = 8$. Part P_0 alone has taken roughly 60 years of CPU-time. Other parts, which were making slow progress were also splitted recursively.

In such a large computation, where thousands of distributed branches of the DPLL-tree have run on hundreds of processors for almost a year, we hope we have not fallen into the trap of an undetected hardware failure (an electricity failure is natural and every detected hardware-failure was re-run from the last state of the search), or a file manipulation error on a particular branch which could contain a good partition of the set $\{1, 2, \dots, 349\}$. Of course, also a software error is possible. We welcome interested readers with proper resources to conduct another search to verify our result. However, we have no doubt that $w(2; 3, 19) = 349$ holds true, since not only do we have the completed run of the first author's dedicated solver, but also the data gathered by the second author, using local search algorithms, gives strong evidence: solving for $n = 348$ is very easy, and in all other similar (smaller) cases this sudden change of behaviour has been validated as been caused by unsatisfiability. We would also (again) like to point out that it seems that there is no alternative method available, and perhaps there will never be. SAT solvers have a long and distinguished history, and the connections to formal methods and verification gives a strong background for correct and efficient implementations. Definitely no less than for example computer algebra systems, but likely more so due to the non-numerical and inherently simple nature of the task. See Section 5 for further remarks on SAT solving for these instances in general. Finally, we remark that it would be highly desirable to be able to substantially compress the resolution proofs obtained from the solver runs, so that a proof object would be obtained which could be verified by certified software (and hardware); see [10] for some recent literature.

2.2. Some new conjectures

In this section, we provide conjectured values of $w(2; 3, t)$ for $t = 20, 21, \dots, 30$. We have used the UbcSAT suite [49] of local-search based satisfiability algorithms for generating good partitions as a proof of the lower bounds we provide. See Section 5 for details of the algorithms used to find the good partitions. Since local search based algorithms are incomplete (they may fail to deliver a satisfying assignment, and hence a good partition when there exists one), it remains to prove exactness of the numbers using a complete satisfiability solver or some complete colouring algorithm.

2.2.1. $w(2; 3, 20) \geq 389$

$$1^{19}0^{11}0^{14}0^{17}0^{10}0^{14}0^{13}0^{19}0^{14}0^{10}0^{14}0^{21}0^{13}0^{10}0^{18}0^{19}0^{14}0^{10}0^{15}0^{13}0^{10}0^{16}0^{18}0^{16}0^{10}0^{13}0^{15}0^{10}0^{14}0^{19} \\ 0^{18}0^{19}0^{10}0^{21}0^{14}0^{10}0^{14}0^{19}0^{13}0^{14}0^{10}0^{17}0^{14}0^{11}0^{20}$$

2.2.2. $w(2; 3, 21) \geq 416$

$$\begin{aligned} &1^8 01^{17} 0101^6 0101^9 01^{12} 0^2 1^{19} 01^{18} 01^3 0101^2 01^{12} 0^2 1^5 0^2 101^{10} 01^{18} 01^3 010^2 1^8 01^{12} 01^{20} 01^{18} 01^3 0101^7 01^8 01^{12} 0^2 \\ &101^{17} 01^{18} 01^3 010^2 1^8 01^5 01^6 0^2 1^{12} 01^6 01^{15} 0101^4 0101^{11} 01^{21} \end{aligned}$$

2.2.3. $w(2; 3, 22) \geq 464$

$$\begin{aligned} &1^2 0^2 1^{17} 0^2 1^9 01^{12} 0101^{12} 01^{15} 01^2 01^{10} 01^4 01^7 01^5 01^{12} 01^9 0101^3 01^8 01^3 0101^9 01^{12} 01^5 01^7 01^4 01^{10} 01^2 01^{15} 01^{12} \\ &0101^{12} 01^9 0^2 1^{17} 0^2 1^9 01^{12} 0101^{12} 01^{15} 01^2 01^{10} 01^4 01^7 01^5 01^{12} 01^9 0101^3 01^8 01^3 01^9 01^{12} 01^5 01^7 01^{11} \end{aligned}$$

2.2.4. $w(2; 3, 23) \geq 516$

$$\begin{aligned} &1^4 0^2 1^2 01^{17} 01^4 0101^{15} 01^{16} 01^4 01^5 01^{20} 0101^2 01^8 0^2 101^4 01^{15} 01^2 01^4 01^{16} 01^9 01^{10} 0101^9 01^{10} 01^7 01^{17} 01^6 0101^{19} 01^{16} 0 \\ &1^{21} 0^2 1^{19} 01^6 01^2 01^{12} 010^2 1^4 0101^{20} 01^{13} 0^2 1^{11} 01^9 0^2 1^6 01^4 01^{13} 0101^3 01^8 01^9 01^{20} 01^5 01^{18} 01^3 01 \end{aligned}$$

2.2.5. $w(2; 3, 24) \geq 593$

$$\begin{aligned} &1^{21} 01^{18} 01^{16} 01^4 01^7 01^6 0101^{14} 01^3 0^2 1^8 01^7 01^3 01^2 01^{20} 0^2 1^3 01^7 01^{15} 01^7 01^3 0^2 1^{20} 01^2 01^3 01^7 01^9 01^{18} 0101^6 01^{21} 01^7 0 \\ &1^{10} 01^7 01^{21} 01^6 0101^{18} 01^9 01^7 01^3 01^2 01^{20} 0^2 1^3 01^7 01^{15} 01^7 01^3 0^2 1^{20} 01^2 01^3 01^7 01^8 0^2 1^3 01^{14} 0101^6 01^7 01^4 01^{16} 01^{18} 01^{21} \end{aligned}$$

2.2.6. $w(2; 3, 25) \geq 656$

$$\begin{aligned} &1^{16} 01^2 01^{19} 01^8 01^7 01^{19} 01^8 01^4 0101^5 01^{17} 01^{10} 01^{21} 0^2 1^2 0101^{10} 01^7 01^{12} 01^4 01^3 01^6 01^7 01^{11} 01^3 01^{13} 01^4 01^{17} 0101^3 \\ &01^6 0^2 1^6 01^{17} 01^8 01^7 01^{13} 01^{14} 01^2 01^4 01^{13} 0^2 1^8 01^7 01^{19} 01^8 01^4 0101^7 01^{15} 01^{10} 01^9 01^{11} 0^2 1^2 0101^{10} 01^5 01^{14} 0 \\ &1^4 01^3 01^6 01^7 01^{11} 01^3 01^{13} 01^4 01^{17} 0101^3 01^6 0^2 1^{24} 01^8 01^9 \end{aligned}$$

2.2.7. $w(2; 3, 26) \geq 727$

$$\begin{aligned} &1^{10} 01^{23} 01^{10} 0101^{20} 01^4 01^{11} 01^6 01^{11} 0^2 1^2 01^5 01^6 01^5 01^{23} 01^4 0101^7 01^{17} 01^{16} 0101^{11} 0^2 1^2 0101^3 01^4 01^2 01^{18} \\ &01^3 01^5 01^{14} 01^{12} 01^{16} 01^4 01^{19} 01^8 010^2 1^4 01^{13} 01^{14} 0101^{20} 01^4 01^{18} 01^{11} 0^2 1^2 01^5 01^6 01^5 01^{23} 01^4 0101^7 01^{15} 0101^{16} \\ &0101^{11} 0^2 1^2 0101^3 01^4 01^2 01^{18} 01^9 01^{14} 01^{12} 01^{16} 01^4 01^{19} 01^8 01^2 01^{18} 01^3 01^{25} \end{aligned}$$

2.2.8. $w(2; 3, 27) \geq 770$

$$\begin{aligned} &1^{24} 01^3 01^5 01^{18} 01^{17} 0101^2 01^{21} 01^3 01^7 01^2 01^{20} 0^2 1^{12} 01^{15} 01^{10} 01^{11} 01^3 01^9 01^6 01^{13} 01^{22} 01^3 0^2 \\ &1^8 01^5 01^{20} 01^6 0101^{16} 01^7 01^3 01^5 010^2 101^{21} 01^2 0^2 1^{14} 01^9 01^{17} 0101^2 01^3 01^{17} 01^3 01^5 0101^2 01^{20} 0^2 1^{12} 01^{15} 01^{22} \\ &01^3 01^9 01^6 01^{13} 01^{22} 01^3 0^2 1^8 01^5 01^{20} 01^8 01^{16} 01^7 01^3 01^5 010^2 101^{24} 01^6 01^8 01^{20} 01^{15} 01^{16} 01^5 \end{aligned}$$

2.2.9. $w(2; 3, 28) \geq 827$

$$\begin{aligned} &1^{27} 01^{10} 01^{22} 0101^{16} 01^{13} 01^{16} 01^{20} 01^4 01^{16} 0101^{11} 0^2 1^2 0101^3 01^6 0^2 1^{18} 01^3 01^5 01^{14} 01^{12} 01^{21} 01^{14} 0 \\ &1^{13} 010^2 1^4 01^{23} 01^4 0101^{25} 01^{18} 01^{11} 01^3 0101^3 01^4 01^7 01^{17} 01^{10} 0101^7 01^{17} 01^{11} 01^4 01^{13} 0^2 1^2 0101^3 01^7 \\ &01^{18} 01^3 01^5 01^{14} 01^{12} 01^{16} 01^4 01^{14} 01^{13} 010^2 1^{18} 01^9 01^4 0101^{25} 01^{18} 01^{11} 01^9 01^4 0101^{23} 01^{10} 01^9 01^{12} 01^{16} 01^{10} \end{aligned}$$

2.2.10. $w(2; 3, 29) \geq 868$

$$\begin{aligned} &1^{15} 01^{21} 01^{18} 01^{17} 01^{18} 01^{21} 01^2 01^7 01^{25} 0101^{12} 01^{17} 01^6 01^7 0^2 1^2 01^{17} 01^5 0^2 101^{10} 01^6 01^{13} 0101^2 01^{17} 01^{16} 01^7 01^4 \\ &01^{20} 0101^2 01^5 01^{11} 01^{25} 01^{10} 01^{25} 01^2 0^2 1^{16} 01^3 01^{10} 01^4 01^{20} 0101^2 01^6 01^{20} 01^{14} 0^2 1^2 01^{17} 01^5 0^2 101^{10} 01^6 01^{13} 0101^2 \\ &01^9 01^7 01^9 01^6 01^7 01^4 01^{20} 0101^2 01^{17} 01^{25} 01^2 01^7 01^{25} 01^3 01^{16} 01^6 01^{12} 01^{25} 01^6 01^{27} 01^8 \end{aligned}$$

2.2.11. $w(2; 3, 30) \geq 903$

$$\begin{aligned} &1^{22} 01^{16} 01^{22} 01^{26} 01^7 0101^8 0101^{22} 01^{12} 01^{16} 01^7 01^6 01^9 01^{11} 01^6 0^2 1^{15} 0^2 1^{13} 01^7 01^8 0101^{23} 01^6 01^{28} 01^7 01^3 01^4 01^{22} \\ &01^8 0101^2 0^2 1^{11} 0101^{22} 01^{11} 01^{14} 01^3 01^{19} 01^4 01^{16} 0^2 1^{11} 0101^5 01^{28} 01^6 0^2 101^{10} 01^2 01^{14} 01^7 01^{10} 01^{23} 01^6 01^{28} \\ &01^7 01^5 01^2 0^2 1^{18} 01^2 01^8 0101^3 01^{11} 0101^{16} 01^{25} 0101^4 01^{23} 01^{16} 01^4 0^2 1^{13} 01^{12} 01^3 01^{27} 01^{10} 01^{14} \end{aligned}$$

We observe that for $t = 24, 25, \dots, 30$ we have $w(2; 3, t) > t^2$, which refutes the assumption that $w(2; 3, t) \leq t^2$, as suggested in [8], based on the exact values for $5 \leq t \leq 16$ known by then.

2.3. Further lower bounds

2.3.1. $w(2; 3, 31) > 930$

$$\begin{aligned} &1^{12}01^{19}01^{16}01^{10}01^901^301^401^{14}01^{12}01^{14}01^{18}0^21^{10}01^401^{13}01^801^50101^{10}01^{17}01^{15}01^{16}01^801^501^80101^401^801^{19} \\ &01^{16}01^501^{18}01^401^{25}01^{14}01^{16}01^301^901^601^{13}01^{11}01^201^{11}01^{13}01^601^901^301^{16}01^{14}01^{25}01^401^{18}01^501^{16}01^{19}01^8 \\ &01^40101^801^501^801^{16}01^{15}01^{17}01^{10}0101^501^801^{13}01^401^{10}0^21^{18}01^{14}01^{12}01^{14}01^401^301^901^{10}01^{16}01^{19}01^{12} \end{aligned}$$

2.3.2. $w(2; 3, 32) > 1006$

$$\begin{aligned} &1^{15}01^{26}01^501^{11}01^401^{14}01^401^{16}01^{28}01^{22}01^201^501^701^{19}01^20101^{16}01^{13}01^501^301^701^401^{13}01^901^{14}01^{28}01^701^{15}01^{10} \\ &010^21^{18}0^21^201^{13}01^601^{21}01^{27}01^20101^801^701^{14}01^{13}01^201^{24}01^{10}01^50101^{17}01^{10}0^21^401^{28}01^901^{11}0^21^801^601^70 \\ &1^{11}01^301^{10}010^21^{28}01^{14}01^{17}01^301^{12}01^{13}01^501^{16}01^{13}01^301^{10}01^{22}01^201^{12}01^701^{28}01^{20}01^201^501^401^{22}01^801^{28} \end{aligned}$$

2.3.3. $w(2; 3, 33) > 1063$

$$\begin{aligned} &1^{29}01^601^{14}01^{11}01^{21}01^{14}01^201^{18}0101^{15}01^{12}01^{25}01^{16}0101^{11}0^21^401^{10}01^501^{13}01^{16}01^{20}01^201^{13}01^{22}01^501^40101^{19}01^90 \\ &1^401^901^{15}0101^{18}01^{11}0^21^401^301^401^201^{28}01^601^701^{29}01^401^{19}01^{10}0^21^{18}01^{14}0101^501^{14}01^{16}01^{28}01^601^{20}01^801^401^7 \\ &01^{14}01^401^{18}01^{11}01^901^401^201^401^{17}0101^301^{14}01^{29}01^{18}0^21^201^{10}01^201^{19}01^201^{10}01^{27}01^{18}01^{12}01^401^{20}01^901^{24}01^{13} \end{aligned}$$

2.3.4. $w(2; 3, 34) > 1143$

$$\begin{aligned} &1^{32}01^70101^801^{29}01^{23}01^30^21^{10}01^301^{17}01^201^901^501^{16}01^{15}01^{20}0^21^201^80^2101^{31}01^{18}01^601^{12}01^601^{17}01^301^70^21^{26}01^6 \\ &01^20101^{24}0^21^701^201^301^{21}01^{24}01^{10}01^{11}01^{20}01^{10}01^{18}01^60^21^{28}01^701^301^{14}01^{27}010^21^{20}0101^401^701^201^{20}0^21^{28}01^701^{18}0 \\ &1^901^{10}01^801^{28}01^5010^2101^301^{27}01^201^501^{32}01^{24}01^601^{25}01^{10}01^201^{23}01^20101^901^{13}01^{10}01^{11}01^{20}01^{12}01^{16}01^701^301^{34} \end{aligned}$$

2.3.5. $w(2; 3, 35) > 1204$

$$\begin{aligned} &1^{34}01^{24}01^801^{22}01^30^21^{10}01^{29}01^701^401^{14}01^{15}01^701^{21}01^601^501^{13}01^{22}0101^{12}01^201^{12}01^701^{11}0^21^401^301^701^{18}01^501^301^40 \\ &1^201^601^{34}01^{19}01^80101^501^{17}01^{10}01^201^{18}01^301^{18}0101^{29}01^{18}01^301^{32}0^21^201^501^{27}01^{16}01^{24}01^301^{12}0101^{17}01^80101^{20}01^{13}01^{22} \\ &01^801^601^701^{21}01^{12}01^{19}01^{11}01^{22}0101^80101^501^{13}01^{14}01^901^{15}0101^{10}01^701^{15}01^{13}01^401^{13}01^{24}01^{12}01^{21}01^{23}01^60^21^{24}01^{35} \end{aligned}$$

2.3.6. $w(2; 3, 36) > 1257$

$$\begin{aligned} &1^{10}01^{33}01^{16}01^{12}01^601^{11}01^{25}0101^401^501^201^{12}01^601^{29}0101^401^{17}01^{26}01^{21}0101^{12}01^30101^20^21^{13}01^501^{16}01^{32}01^201^{19}0^210 \\ &1^401^301^{25}01^{10}01^{32}01^501^201^{14}01^701^{10}01^{17}01^{16}01^401^{20}01^{16}01^{13}01^501^{12}01^301^{11}0101^{34}01^901^{10}01^{30}01^301^501^{26}01^901^{11}0 \\ &1^601^201^{33}01^201^{10}01^201^{20}01^601^301^{24}01^{16}01^{19}0^2101^401^301^{17}01^701^{16}01^{21}01^{26}0101^501^{16}01^{26}01^301^40^21^{13}01^401^201^8010 \\ &1^{22}01^{16}01^{20}01^{28}0101^{34} \end{aligned}$$

2.3.7. $w(2; 3, 37) > 1338$

$$\begin{aligned} &1^501^{33}01^{22}01^201^{12}0^21^{18}01^{14}01^901^{12}01^{15}01^{24}01^{19}01^{17}01^70^21^70101^{12}01^201^{10}01^{22}01^801^{32}01^301^601^{19}01^{14}01^{21}0^21^201^{12} \\ &01^701^{26}01^{22}01^20101^301^{32}01^301^{14}01^{11}01^{17}01^{18}0^21^20101^301^701^{22}01^{32}01^{22}0101^{28}0^21^401^{31}01^601^{17}01^301^{14}01^{21}01^401^7 \\ &01^{14}01^{35}01^501^{16}01^{19}01^301^40101^{23}01^{10}01^{31}01^{12}0101^{11}01^301^{12}01^501^{14}01^201^{10}01^{27}01^{30}0^21^20101^801^201^401^{27}010^21^70 \\ &1^{29}01^401^{14}01^{16}01^{18}01^{14}01^{21}0^21^{11}01^6 \end{aligned}$$

2.3.8. $w(2; 3, 38) > 1378$

$$\begin{aligned} &1^{34}01^{14}01^701^{13}01^{22}0101^{12}01^{23}01^{12}01^{13}010^21^{28}01^401^201^{24}01^{10}01^701^{29}01^401^{19}01^801^201^{18}0^21^{21}01^{14}01^{13}01^701^{17} \\ &0^21^401^{26}01^30101^{10}01^{20}01^{16}01^{18}01^901^{12}0101^{21}01^601^{37}01^601^{15}01^{13}01^601^{12}01^801^{32}01^401^501^{19}01^301^{10}01^{21}0^21^401^{18} \\ &01^{28}01^{37}01^601^{11}01^301^{10}01^701^{13}0^21^401^{34}01^501^401^{25}01^50^21^301^{27}01^{10}01^501^{23}01^401^{22}01^801^{12}01^{16}01^{13}0^21^401^{20}0 \\ &1^{10}01^{33}01^70101^{14}01^501^{27}01^70101^60^21^{19}01^{16}01^{31} \end{aligned}$$

2.3.9. $w(2; 3, 39) > 1418$

1²01⁴01¹³01³⁴0101²⁰01²¹01¹⁹01⁷01²01³⁶01²⁰01²01⁸01²⁴01¹²01⁵01²⁷01¹⁸01¹¹01⁷01⁶01³⁰01⁵01¹⁶01⁴01²⁶01⁴
0²1²⁸01²01⁴01²01⁶01¹⁸01²⁶01¹⁷01³01⁷01¹⁶01¹¹01⁵01²01²²01³01⁶01⁴01³⁶01²⁴01⁸01¹⁶01⁶01¹³0101³⁴01²⁶01³⁶
01⁶01²01⁵01¹⁶01¹⁵01³01⁹01⁷01¹¹01²01²⁴01¹⁵01²⁷0101²01³01⁹01¹⁷010²1³⁶01²²01¹¹01²⁴01²⁸01⁶01²⁹0²1⁵0²
1³01³²01²0²1⁵01²01¹⁵01⁴01⁷01⁴01²⁹01⁸01²²01⁴01³²01¹²01³¹

2.4. A conjecture on the upper bound

An important theoretical question is the growth-rate of $t \mapsto w(2; 3, t)$. Although the precise relation “ $w(2; r, t) \leq t^2$ ” has been invalidated by our results, quadratic growth still seems appropriate (see [35] for a more general conjecture on polynomial growth for van der Waerden numbers in certain directions of the parameter space):

Conjecture 2.1. There exists a constant $c > 1$ such that $w(2; 3, t) \leq ct^2$.

By observing the known exact values, computed conjectures, and lower bounds, we may arrive at the following recursion:

$$w(2; 3, t) \leq w(2; 3, t-1) + d(t-1),$$

for $t \geq 3$ with $w(2; 3, 3) = 9$. From our data, we observe that $d \geq \frac{593-516}{23} \approx 3.35$. Solving the recursion, we get

$$w(2; 3, t) \leq 1.675(t^2 - t) - 1.05 < 1.675t^2,$$

which satisfies all data regarding $w(2; 3, t)$, presented so far.

3. Patterns in the good partitions

In this section, we investigate the set of all good partitions corresponding to certain van der Waerden numbers $w(2; 3, t)$ for patterns. Let $S(t)$ denote the set of all binary strings each of which represent a good partition of the set $\{1, 2, \dots, w(2; 3, t) - 1\}$. Generating $S(t)$ involves traversing the respective search space completely. The lower bounds reported in the previous section were obtained using local search based satisfiability algorithms, where we stopped after the first satisfying assignment corresponding to a good partition is found. On the other hand, to generate all good partitions, we need to use a complete algorithm to visit the total search space. So we restrict our search for patterns in $3 \leq t \leq 14$ (with exceptions in Table 5, where $t = 15, \dots, 20$ and 24 are also considered).

Let $n_0(B)$, $n_1(B)$, and $n_{00}(B)$ be the number of zeros, ones, and double-zeros, respectively, in a bitstring B (note that three consecutive zeros are not possible in a $B \in S(t)$). Let $\text{EP1S}(B)$ denote the sequence of powers of 1 in a bitstring B . Let $n_p(B)$ and $n_v(B)$ denote the number of peaks (local maxima) and valleys (local minima), respectively, in $\text{EP1S}(B)$ (not necessarily strict). For example, for the compact bitstring $1^8 001^6 01^3 01^1 01^3 001^5 01^8 01^5 001^3 01^1 01^3 01^6 001^8$ (with $n_0 = 16$, $n_1 = 60$ and $n_{00} = 4$), we have the following EP1S, with p and v, marking peaks and valleys, respectively, corresponding to changes in magnitudes.

8 6 3 1 3 5 8 5 3 1 3 6 8
p v p v p

And for $B = 1^1 01^1 01^2 01^2 01^3 01^3$ we have $n_0(B) = 5$, $n_1(B) = 12$, $n_{00}(B) = 0$, while there is one valley followed by one peak, and thus $n_v(B) = n_p(B) = 1$.

3.1. Number of 0's and 00's

In this section, we count the number $\min\{n_0(B) : B \in S(t)\}$, $\max\{n_0(B) : B \in S(t)\}$, and $\max\{n_{00}(B) : B \in S(t)\}$ for $3 \leq t \leq 14$. Observations in Table 1 lead us to Conjectures 3.1 and 3.2.

Table 1: Zeros in good-partitions of $\{1, 2, \dots, w(2; 3, t) - 1\}$

$w(2; 3, t)$	$\min\{n_0(B) : B \in S(t)\},$ $\max\{n_0(B) : B \in S(t)\}$	$\max\{n_{00}(B) : B \in S(t)\}$
$w(2; 3, 3)$	(4, 4)	2
$w(2; 3, 4)$	(6, 6)	2
$w(2; 3, 5)$	(7, 9)	2
$w(2; 3, 6)$	(8, 10)	4
$w(2; 3, 7)$	(11, 12)	3
$w(2; 3, 8)$	(14, 14)	1
$w(2; 3, 9)$	(16, 16)	4
$w(2; 3, 10)$	(19, 21)	5
$w(2; 3, 11)$	(19, 22)	5
$w(2; 3, 12)$	(22, 22)	1
$w(2; 3, 13)$	(25, 29)	5
$w(2; 3, 14)$	(29, 29)	4

Conjecture 3.1. $|n_0(B) - n_0(B')| \leq t, \forall B, B' \in S(t)$ with $t \geq 3$.

Conjecture 3.2. $n_{00}(B) < t, \forall B \in S(t)$ with $t \geq 3$.

3.2. Number of 1's

In this section, we count $T = \min\{n_p(\text{EPIS}(B)) + n_v(\text{EPIS}(B)) : B \in S(t)\}$, as well as minimum and maximum values of $n_1(B)$ over all $B \in S(t)$. The observations in Table 2 lead us to Conjecture 3.3, and Questions 3.1 and 3.2.

Table 2: Selected good-partitions of $\{1, 2, \dots, w(2; 3, t) - 1\}$

$w(2; 3, t)$	A good partition B corresponding to T	T	$\min\{n_1(B) : B \in S(t)\},$ $\max\{n_1(B) : B \in S(t)\}$
$w(2; 3, 3)$	$1^2 001^2 00$ (2 2)	1	(4, 4)
$w(2; 3, 4)$	$1^3 001^1 01^3 001^1 01^3$ (3 1 3 1 3)	5	(11, 11)
$w(2; 3, 5)$	$001^3 001^1 01^4 001^4 01^1$ (3 1 4 4 1)	4	(12, 14)
$w(2; 3, 6)$	$01^5 001^5 01^3 001^5 001^5$ (5 5 3 5 5)	3	(21, 23)
$w(2; 3, 7)$	$1^1 01^1 01^4 01^2 01^5 01^4 01^1 001^3 01^5 01^2 01^5 0$ (1 1 4 2 5 4 1 3 5 2 5)	8	(33, 34)
$w(2; 3, 8)$	$1^4 01^2 01^4 01^1 01^4 01^3 01^5 001^5 01^3 01^4 01^1 01^4 01^2 01^1$ (4 2 4 1 4 3 5 5 3 4 1 4 2 1)	12	(43, 43)
$w(2; 3, 9)$	$1^8 001^6 01^3 01^1 01^3 001^5 01^8 01^5 001^3 01^1 01^3 01^6 001^8$ (8 6 3 1 3 5 8 5 3 1 3 6 8)	5	(60, 60)
$w(2; 3, 10)$	$1^7 01^4 01^2 01^5 001^2 001^7 01^4 01^8 01^1 01^8 01^4 001^6 001^2 001^8 01^9$ (7 4 2 5 2 7 4 8 1 8 4 6 2 8 9)	13	(75, 77)
$w(2; 3, 11)$	$01^{10} 01^4 001^6 01^{10} 01^2 001^9 01^6 01^1 01^9 001^1 001^{10} 01^6 001^{10} 01^{10}$ (10 4 6 10 2 9 6 1 9 1 10 6 10 10)	11	(91, 94)
$w(2; 3, 12)$	$1^9 01^8 01^9 01^2 01^3 01^1 01^7 01^2 0101^3 01^{11} 0^2$ $1^{11} 01^3 0101^2 01^7 01^1 01^3 01^2 01^9 01^8 01^9$ (9 8 9 2 3 1 7 2 1 3 11 11 3 1 2 7 1 3 2 9 8 9)	17	(112, 112)
$w(2; 3, 13)$	$1^1 01^6 01^{12} 01^4 001^{11} 001^6 01^{10} 01^2 01^4 01^{11} 01^1 0$ $1^6 01^9 01^2 01^3 01^7 01^{10} 01^1 001^5 01^{12} 01^5 01^4 01^2$ (1 6 12 4 11 6 10 2 4 11 1 6 9 2 3 7 10 1 5 12 5 4 2)	15	(130, 134)

Conjecture 3.3. $n_1(B') \leq n_1(B) \leq w(2; 3, t-1) + 2t$, $\forall B \in S(t), B' \in S(t-1)$.

Question 3.1. For each positive constant c does there exist a t' such that for all $t \geq t'$, $n_p(\text{EP1S}(B)) + n_v(\text{EP1S}(B)) \geq ct$, ($t \geq 3$) $\forall B \in S(t)$? (We conjecture yes).

Question 3.2. Is there a good partition $B \in S(t)$, ($t \geq 4$) with 3 consecutive numbers equal in $\text{EP1S}(B)$? (Note that, for $t = 3$, the partition $1^1 01^1 001^1 01^1$ has four consecutive exponents, which are the same.)

4. Palindromes

Recall:

1. for given $k \in \mathbb{N}$ (the number of “colours”),
2. t_0, \dots, t_{k-1} (the lengths of arithmetic progressions),
3. and $n \in \mathbb{N}$ (the number of vertices)

we consider block partitions (P_0, \dots, P_{k-1}) of $\{1, \dots, n\}$ (that is, $P_0 \cup \dots \cup P_{k-1} = \{1, \dots, n\}$ and $P_i \cap P_j = \emptyset$ for $i \neq j$; the P_i may be empty) such that no P_i contains an arithmetic progression of length t_i — these are the “good partitions”, and $w(k; t_0, \dots, t_{k-1}) \in \mathbb{N}$ is the smallest n such that no good partitions exists.³ If (P_0, \dots, P_{k-1}) is a good partition w.r.t. k, t_0, \dots, t_{k-1} and n , then for $1 \leq n' < n$ we obtain a good partition w.r.t. k, t_0, \dots, t_{k-1} and n' from P by just removing vertices $n' + 1, \dots, n$ from their blocks. Thus $w(k; t_0, \dots, t_{k-1})$ completely determines for which $n \in \mathbb{N}$ good partitions exist, namely exactly for $n < w(k; t_0, \dots, t_{k-1})$.

Now if (P_0, \dots, P_{k-1}) is a good partition w.r.t. n , then also (P'_0, \dots, P'_{k-1}) is a good partition w.r.t. n , where every vertex $v \in \{1, \dots, n\}$ is replaced by $v' := n+1-v$. So it is of interest to consider self-symmetric partitions (with $P' = P$), and we call such partitions *good palindromic partitions*, motivated by the fact that a partition as a string of numbers from $\{0, \dots, k-1\}$ is palindromic iff the string is a palindrome (reads the same forwards and backwards). For example, $01^2 001^2 0$ represents a good palindromic partition for $k = 2$, $t_0 = t_1 = 3$ and $n = 8$ (namely $(\{1, 4, 5, 8\}, \{2, 3, 6, 7\})$), and so does $(\{1, 3, 6, 8\}, \{2, 4, 5, 7\})$, represented by $0101^2 010$, while $(\{1, 2, 5, 6\}, \{3, 4, 7, 8\})$, represented by $001^2 001^2$, is a good partition which is not palindromic.

For given k and t_0, \dots, t_{k-1} again we want to completely determine (in theory) for which n do good palindromic partitions exist and for which not. The key is the following observation (with obvious proof; it will follow also from Lemmas 4.2, 4.3).

Lemma 4.1. Consider fixed k, t_0, \dots, t_{k-1} , and $n \geq 3$. From a good palindromic partition (P_0, \dots, P_{k-1}) w.r.t. n we obtain a good palindromic partition (P'_0, \dots, P'_{k-1}) w.r.t. $n-2$ by removing vertices $1, n$ and replacing the remaining vertices v by $v-1$, that is, $P'_i := \{v-1 : v \in P_i \setminus \{1, n\}\}$.

Corollary 4.1.1. If for n there is no good palindromic partition, then there is no good palindromic partition for all $n+2 \cdot i$, $i \in \mathbb{N}_0$.

Since by van der Waerden’s theorem we know there always exists some n such that for all $n' \geq n$ no good palindromic partition exists, we get that the existence of good palindromic partitions w.r.t. fixed t_0, \dots, t_{k-1} is determined by two numbers, the endpoint p of “always exists” resp. q of “never exists”, with alternating behaviour in the interval in-between:

Corollary 4.1.2. Consider the maximal $p \in \mathbb{N}_0$ such that for all $n \leq p$ good palindromic partitions exist, and the minimal $q \in \mathbb{N}$ such that for all $n \geq q$ no good palindromic partitions exist. Then $q-p$ is an odd natural number, where no good palindromic partitions exists for $p+1$, but $p+2$ again has a good palindromic partition, and so on alternatingly, until from q on no good palindromic partitions exist anymore.

³ $\mathbb{N} = \{1, \dots\}$, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$

Definition 4.1. The *palindromic van-der-Waerden number* $\text{pdw}(k; t_0, \dots, t_{k-1}) \in \mathbb{N}_0^2$ is defined as the pair (p, q) such that p is the largest $p \in \mathbb{N}_0$ with the property that for all $1 \leq n \leq p$ there exists a good palindromic partition w.r.t. n , while q is the smallest $q \in \mathbb{N}$ such that for no $n \geq q$ there exists a good palindromic partition w.r.t. n . We use $\text{pdw}(k; t_0, \dots, t_{k-1})_1 = p$ and $\text{pdw}(k; t_0, \dots, t_{k-1})_2 = q$. So $0 \leq \text{pdw}(k; t_0, \dots, t_{k-1})_1 < \text{pdw}(k; t_0, \dots, t_{k-1})_2 \leq w(k; t_0, \dots, t_{k-1})$.

The *palindromic gap* is

$$\text{pdg}(k; t_0, \dots, t_{k-1}) := w(k; t_0, \dots, t_{k-1}) - \text{pdw}(k; t_0, \dots, t_{k-1})_2 \in \mathbb{N}_0,$$

while the *palindromic span* is defined as

$$\text{pds}(k; t_0, \dots, t_{k-1}) := \text{pdw}(k; t_0, \dots, t_{k-1})_2 - \text{pdw}(k; t_0, \dots, t_{k-1})_1 \in \mathbb{N}.$$

To certify that $w(k; t_0, \dots, t_{k-1}) = n$ holds means to show that there exists a good partition for $n - 1$ w.r.t. (t_0, \dots, t_{k-1}) and to show that there is no good partition for n w.r.t. (t_0, \dots, t_{k-1}) . For palindromic numbers we need to double the effort:

Corollary 4.1.3. To certify that $\text{pdw}(k; t_0, \dots, t_{k-1}) = (p, q)$ holds, exactly the following needs to be shown:

- there are good palindromic partitions for $p - 1$ and $q - 1$ w.r.t. (t_0, \dots, t_{k-1}) ;
- there are no good palindromic partitions for $p + 1$ and $q + 1$ w.r.t. (t_0, \dots, t_{k-1}) .

4.1. Palindromic vdW-hypergraphs

Recall that a finite hypergraph G is a pair $G = (V, E)$, where V is a finite set (of “vertices”) and E is a set of subsets of V (the “hyperedges”); one writes $V(G) := V$ and $E(G) := E$. The essence of the (finite) van der Waerden problem is given by the hypergraphs $\text{ap}(t, n)$ of arithmetic progressions with progression length $t \in \mathbb{N}$ and the number $n \in \mathbb{N}_0$ of vertices (where the arithmetic progressions of length t in \mathbb{N} are the subsets of the form $\{a + i \cdot d\}_{i \in \{0, \dots, t-1\}}$ for $a, d \in \mathbb{N}$):

- $V(\text{ap}(t, n)) := \{1, \dots, n\}$
- $E(\text{ap}(t, n)) := \{p \subseteq \{1, \dots, n\} : p \text{ arithmetic progression of length } t\}.$

For example $\text{ap}(3, 5) = (\{1, 2, 3, 4, 5\}, \{\{1, 2, 3\}, \{2, 3, 4\}, \{1, 3, 5\}, \{3, 4, 5\}\})$. The diagonal vdW-number $w(k; t, \dots, t)$ for $k, t \in \mathbb{N}$ now is the smallest $n \in \mathbb{N}$ such that the hypergraph $\text{ap}(t, n)$ is not k -colourable, where in general a k -colouring of a hypergraph G is a map $f : V(G) \rightarrow \{1, \dots, k\}$ such that no hyperedge is “monochromatic”, that is, every hyperedge gets at least two different values by f . We see that the SAT-encoding of “ $w(2; 3, t) > n$?” as discussed in Subsection 1.1 exactly consists of the two hypergraphs $\text{ap}(3, n)$ and $\text{ap}(t, n)$ represented by positive resp. negative clauses.

The task now is to define the palindromic version $\text{pdap}(t, n)$ of the hypergraph of arithmetic progressions, so that for diagonal palindromic vdW-numbers we have that $\text{pdw}(k; t, \dots, t)$ is the smallest $n \in \mathbb{N}$ such that the hypergraph $\text{pdap}(t, n)$ is not k -colourable, and such that for two-coloured problems (i.e., $k = 2$) the SAT-encoding of “ $\text{pdw}(2; t_0, t_1) > n$?” (satisfiable iff the answer is yes) consists exactly of the two hypergraphs $\text{pdap}(t_0, n)$, $\text{pdap}(t_1, n)$ represented by positive resp. negative clauses (while for more than two colours generalised clause-sets can be used; see [36]).

Consider fixed $t \in \mathbb{N}$ and $n \in \mathbb{N}_0$. Obviously $\text{pdap}(t, 0) := \text{ap}(t, 0) = (\{\}, \{\})$, and so assume $n \geq 1$. Let the permutation m (like “mirror symmetry”) of $\{1, \dots, n\}$ be the map $v \mapsto v' := n + 1 - v$ as discussed in the introduction. As every permutation, m induces an equivalence relation \sim on $\{1, \dots, n\}$ by considering the cycles, which here, since m is an involution (self-inverse), just has the equivalence classes $\{1, \dots, n\}/\sim = \{\{v, f(v)\}\}_{v \in \{1, \dots, n\}}$ of size 1 or 2 comprising the elements and their images. Note that m has a fixed point (an equivalence class of size 1) iff n is odd, in which case the unique fixed point is $\frac{n+1}{2}$. The idea now is to define $m' : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, which chooses from each equivalence class one representative (so $m'(v) \in \{v, m(v)\}$ and $v \sim w \Leftrightarrow m'(v) = m'(w)$), and to let $\text{pdap}(t, n)$ be the image of $\text{ap}(t, n)$ under m' , that is, $(m'(V(\text{ap}(t, n))), \{m'(H)\}_{H \in E(\text{ap}(t, n))})$. Naturally we choose $m'(v)$ to be the smaller of v and $m(v)$. Now it occurs that images of arithmetic progressions under m' can subsume each other, and so we define $\text{pdap}(t, n)$ as the image of $\text{ap}(t, n)$ under m' , where also all subsumed hyperedges are removed (so we only keep the minimal hyperedges under the subset-relation).

Definition 4.2. For $t \in \mathbb{N}$ and $n \in \mathbb{N}_0$ the hypergraph $\text{pdap}(t, n)$ is defined as follows:

- $V(\text{pdap}(t, n)) := \{1, \dots, \lceil \frac{n}{2} \rceil\}$
- $E(\text{pdap}(t, n))$ is the set of minimal elements w.r.t. \subseteq of the set of $m'_n(H)$ for $H \in E(\text{ap}(t, n))$, where $m'_n : \{1, \dots, n\} \rightarrow V(\text{pdap}(t, n))$ is defined by $m'_n(v) := v$ for $v \leq \lceil \frac{n}{2} \rceil$ and $m'_n(v) := n + 1 - v$ for $v > \lceil \frac{n}{2} \rceil$.

Using $\text{ap}(3, 5) = (\{1, 2, 3, 4, 5\}, \{\{1, 2, 3\}, \{2, 3, 4\}, \{1, 3, 5\}, \{3, 4, 5\}\})$ as above, we have $m'(\{1, 2, 3\}) = \{1, 2, 3\}$, $m'(\{2, 3, 4\}) = \{2, 3\}$, $m'(\{1, 3, 5\}) = \{1, 3\}$ and $m'(\{3, 4, 5\}) = \{1, 2, 3\}$, whence $\text{pdap}(3, 5) = (\{1, 2, 3\}, \{\{1, 3\}, \{2, 3\}\})$.

Lemma 4.2. Consider $t \in \mathbb{N}$ and $n \in \mathbb{N}_0$. The hypergraph $\text{pdap}(t, n)$ is embedded into the hypergraph $\text{pdap}(t, n + 2)$ via the map $e : V(\text{pdap}(t, n)) \rightarrow V(\text{pdap}(t, n + 2))$ given by $v \mapsto v + 1$.

Proof. First we note that $|V(\text{pdap}(t, n + 2))| = |V(\text{pdap}(t, n))| + 1$, and so the range of e is $V(\text{pdap}(t, n + 2)) \setminus \{1\}$. Let G be the hypergraph with vertex set $V(\text{pdap}(t, n + 2)) \setminus \{1\}$, whose hyperedges are all those hyperedges $H \in E(\text{pdap}(t, n + 2))$ with $1 \notin H$. We show that e is an (hypergraph-)isomorphism from $\text{pdap}(t, n)$ to G , which proves the assertion.

Now obviously the underlying hypergraph $\text{ap}(t, n)$ is embedded into the underlying $\text{ap}(t, n + 2)$ via the underlying map $v \in V(\text{ap}(t, n)) \mapsto v + 1 \in V(\text{ap}(t, n + 2))$, where the image of this embedding is given by the hypergraph with vertex set $V(\text{ap}(t, n + 2)) \setminus \{1, n + 2\}$, and where the hyperedges are those $H \in E(\text{ap}(t, n + 2))$ with $1, n + 2 \notin H$. Since $m'_{n+2}(n + 2) = 1$ and $m'_n(v) = m'_{n+2}(v + 1) - 1$ for $v \in \{1, \dots, n\}$, the assertion follows from the fact that there are no hyperedges $H, H' \in E(\text{ap}(t, n + 2))$ with $H \cap \{1, n + 2\} \neq \emptyset$, $H' \cap \{1, n + 2\} = \emptyset$ and $m'_{n+2}(H) \subset m'_{n+2}(H')$ (thus $m'_{n+2}(H')$ can only be removed from $\text{pdap}(t, n + 2)$ by subsumptions already at work in $\text{pdap}(t, n)$), and this is trivial since $1 \in m'_{n+2}(H)$ but $1 \notin m'_{n+2}(H')$. \square

The SAT-translation of “ $\text{pdw}(2; t_0, t_1) > n$?” is accomplished similar to the translation of “ $w(2; t_0, t_1) > n$?”, now using $\text{pdap}(t_0, n)$, $\text{pdap}(t_1, n)$ instead of $\text{ap}(t_0, n)$, $\text{ap}(t_1, n)$:

Lemma 4.3. Consider $t_0, t_1 \in \mathbb{N}$, $t_0 \leq t_1$, and $n \in \mathbb{N}_0$. Let the boolean clause-set $F^{\text{pd}}(t_0, t_1, n)$ be defined as follows:

- the variable-set is $\{1, \dots, \lceil \frac{n}{2} \rceil\}$ ($= V(\text{pdap}(t_0, n)) = V(\text{pdap}(t_1, n))$);
- the hyperedges of $\text{pdap}(t_0, n)$ are directly used as positive clauses;
- the hyperedges H of $\text{pdap}(t_1, n)$ yield negative clauses $\{\bar{v}\}_{v \in H}$.

Then there exists a good palindromic partition if and only if $F^{\text{pd}}(t_0, t_1, n)$ is satisfiable, where the satisfying assignments are in one-to-one correspondence to the good palindromic partitions of $\{1, \dots, n\}$ w.r.t. (t_0, t_1) . \square

For more than two colours, Lemma 4.3 can be generalised by using generalised clause-sets, as in [36], and there one also finds the “generic translation”, a general scheme to translate generalised clause-sets (with non-boolean variables) into boolean clause-sets (see also [38, 39]).

4.2. Precise values

See Subsection 5.1 for details of the computation.

Table 3: Palindromic vdW-numbers $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t)$	$\text{pds}(2; 3, t)$	$\text{pdg}(2; 3, t)$
3	(6, 9)	3	0
4	(15, 16)	1	2
5	(16, 21)	5	1
Continued on Next Page...			

Table 3: Palindromic vdW-numbers $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t)$	$\text{pds}(2; 3, t)$	$\text{pdg}(2; 3, t)$
6	(30, 31)	1	1
7	(41, 44)	3	2
8	(52, 57)	5	1
9	(62, 77)	15	0
10	(93, 94)	1	3
11	(110, 113)	3	1
12	(126, 135)	9	0
13	(142, 155)	13	5
14	(174, 183)	9	3
15	(200, 205)	5	13
16	(232, 237)	5	1
17	(256, 279)	23	0
18	(299, 312)	13	0
19	(338, 347)	9	2
20	(380, 389)	9	≥ 0
21	(400, 405)	5	≥ 11
22	(444, 463)	19	≥ 1
23	(506, 507)	1	≥ 9
24	(568, 593)	25	≥ 0
25	(586, 607)	21	≥ 49

4.3. Conjectured values and bounds

For $26 \leq t \leq 39$ we have reasonable values on $\text{pdw}(2; 3, t)$, which are given in Table 4, and which we believe to be exact for $t \leq 35$. These values have been computed by local-search methods (see Subsection 5.2), and thus for sure we can only say that they present lower bounds. We obtain conjectured values for the palindromic span (which might however be too large or too small) and conjectured values for the palindromic gap (which additionally depend on the conjectured values from Subsection 2.2, while for $t \geq 31$ we only have the lower bounds from Subsection 2.3).

Table 4: Conjectured palindromic vdW-numbers $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t) \geq$	$\text{pds}(2; 3, t) \sim$	$\text{pdg}(2; 3, t) \sim$
26	(634, 643)	9	84
27	(664, 699)	35	71
Continued on Next Page. . .			

Table 4: Conjectured palindromic vdW-numbers $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t) \geq$	$\text{pds}(2; 3, t) \sim$	$\text{pdg}(2; 3, t) \sim$
28	(728, 743)	15	84
29	(810, 821)	11	47
30	(844, 855)	11	48
31	(916, 931)	15	0
32	(958, 963)	5	44
33	(996, 1005)	9	59
34	(1054, 1081)	27	63
35	(1114, 1155)	41	50
36	(1186, 1213)	27	45
37	(1272, 1295)	23	44
38	(1336, 1369)	33	10
39	(1406, 1411)	5	8

4.4. Certificates

Table 5 gives good palindromic partitions for $n_1 - 1, n_2 - 1$ with $(n_1, n_2) = \text{pdw}(2; 3, t)$, $3 \leq t \leq 39$, according to the values in Tables 3, 4. Due to the palindromic property, for the corresponding n we only show the partition of $\{1, \dots, \lceil \frac{n}{2} \rceil\}$, so that for example the good palindromic partition $01^2 001^2 0$ for $t = 3$ and $n = 8$ is compressed to $01^2 0$. Note that such partitions correspond exactly to the solution of $F^{\text{pd}}(3, t, n)$ as defined in Lemma 4.3.

Table 5: Good palindromic partitions for $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t) - 1$	Good palindromic partitions
3	(5, 8)	$1^2 0, 10^2 1$
4	(14, 15)	$0101^3 0, 1^2 010^2 1^2$
5	(15, 20)	$1^4 0^2 1^2, 1^2 0101^4 0$
6	(29, 30)	$1^2 01^5 0^2 1^3 01, 1^5 0^2 1^5 0^2 1$
7	(40, 43)	$01^4 0101^3 0^2 1^5 01, 0^2 1^4 01^2 01^5 01^4 01$
8	(51, 56)	$1^7 01^2 01^3 0^2 101^4 01^3, 1^2 01^2 01^4 0101^4 01^3 01^5 0$
9	(61, 76)	$1^4 01^3 0^2 1^5 01^2 01^5 01^4 01, 1^8 0^2 1^6 01^3 0101^3 0^2 1^5 01^4$
10	(92, 93)	$1^9 01^4 010^2 1^9 0^2 10^2 1^9 01^4, 1^9 01^8 0^2 1^2 0^2 1^6 0^2 1^4 01^8 01$
11	(109, 112)	$1^3 01^3 01^9 010^2 1^8 0^2 1^3 0101^8 01^4 01^4, 1^2 0101^3 0101^4 01^7 0^2 1^5 01^3 01^6 0101^4 01$
12	(125, 134)	$1^{11} 0101^8 010^2 101^{10} 01^8 01^5 01^4 0101^2, 1^9 01^8 01^9 01^2 01^3 0101^7 01^2 0101^3 01^{11} 0$
13	(141, 154)	$101^2 01^{11} 01^{10} 010^2 1^4 01^{10} 01^{11} 010^2 1^6 01^2, 1^{10} 01^4 01^{11} 01^4 01^{10} 010^2 1^3 0^2 101^{10} 01^4 01^2 01^4$
Continued on Next Page...		

Table 5: Good palindromic partitions for $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t) - 1$	Good palindromic partitions
14	(173, 182)	$1^2 01^7 01^6 01^3 01^7 01^6 01^{10} 01^{10} 01^5 01 01^6 01^2 0^2 1^7, 1^4 0^2 1^2 01^8 01 01^7 01^8 01^5 01^8 01^7 01 01^8 01^2 0^2 1^{13} 01^2$
15	199 204	$1^5 01^{10} 01^7 0^2 1^4 01^2 01^5 01^7 01^8 01^{10} 01^3 01^{12} 01^5 01^2 01^4 01$ $1^{11} 01^6 01^9 01^2 01^{11} 01^3 01^5 01^2 0^2 1^2 01^{11} 01 01^{10} 01^{13} 01^2$
16	231 236	$1^{14} 01^9 01^6 01^5 01^2 0^2 1^4 01^{12} 01^3 01^7 01^2 0^2 1^{12} 01^8 01^5 01^{10} 01$ $1^{15} 01^9 01^2 0^2 1^4 01^9 01 0^2 1^9 01^{14} 01^9 0^2 1^{11} 01^5 01^2 01^9 01^3$
17	255 278	$1^{14} 01^{16} 01 01^2 01^{16} 01 01^5 01^6 01^3 01^9 01^2 01^{11} 01 01^7 01^6 01^3 0^2 1^8$ $1^4 01^6 01^5 01^3 01^{10} 01^6 01^5 01^8 0^2 1^{10} 01 01^6 01^7 01 01^8 01^{12} 01 01^{16} 01^9 0^2 1$
18	298 311	$1^2 01^2 01^{16} 01^6 01 01^{16} 0^2 1^4 01^{17} 01^2 01^5 01 01^{13} 01^2 01^6 01^2 01^5 01^{12} 0^2 1^{10} 01^7$ $1^6 01^8 01^3 01^{17} 0^2 1^3 0^2 1^{12} 01^{16} 01^7 01^9 0^2 10^2 1^4 01^9 01^7 01^4 01^{14} 01^{10} 01^6$
19	337 346	$1^{11} 01^2 01 01^{15} 0^2 1^3 01^2 01^{16} 01^{17} 0^2 101^8 01 0^2 1^{17} 01^{16} 01^2 01^3 0^2 101^{15} 01^2 01^{10} 01^3$ $1^{11} 01^{18} 01^5 01 01^6 01^7 01 01^3 01^2 01^{18} 01^9 01 01^2 0^2 1^{14} 01^3 01^2 01 01^{12} 01^3 01 01^{12} 01^{13} 01^5$
20	379 388	$1^{16} 0^2 101^{10} 01^{18} 01^5 01^6 0^2 1^2 01^5 01^{15} 01^{10} 01 01^{16} 01^3 01 01^{16} 01^2 01^9 0^2 1^{19} 01 01^2 01^9$ $1^{19} 01^{11} 01^4 01^7 01 01^4 01^{13} 01^9 01^4 01 01^{14} 0^2 1^3 0^2 101^9 01^{18} 01^9 01^4 01 01^5 01^3 01^{10} 01^{16} 01^4$
21	399 404	$1^6 01 01^6 0^2 1^{11} 01^3 01^4 01^6 01^{10} 01^{19} 01^7 01^2 01^3 01^{10} 01^6 0^2 1^4 01^6 01^2 01^{12} 01^{16} 0^2 1^{19} 01 01^5 01^8 01^7$ $1^{18} 01 01^5 0^2 1^3 01^{13} 01^9 0^2 1^{19} 01 01^8 01 01^{16} 01^7 01 01^{14} 01^8 01 01^{20} 01 01^3 01^{10} 01^{17} 01$
22	443 462	$1^3 01^8 01^7 01^8 01^{19} 01^8 01^7 01 01^{12} 01^7 01 01^8 01^{16} 0^2 1^{16} 01^5 01^2 01 01^{20} 01 01^{16} 01^{19} 01^3 01^2 01 01^6$ $1^{18} 01^{17} 01^6 01^4 01^7 01 01^2 1^5 01^{15} 01^{10} 01^{14} 0^2 101^6 0^2 1^2 01^8 01^{14} 0^2 1^7 01^4 01^{10} 01^{21} 01^{10} 01^{11} 01^4 0^2 1^9$
23	505 506	$1^{11} 01^{22} 01^{20} 0^2 1^9 01^4 01^8 0^2 1^6 01^{22} 0^2 1^3 0^2 1^6 01^{10} 01^{11} 01^2 01^{11} 01^{10} 01^6 0^2 1^3 0^2 1^{22} 01^6 01^9 01^4 01^{10} 01^{10}$ $1^{22} 0^2 1^{18} 01^2 01^{19} 01^3 01^2 01^8 0^2 1^{18} 01^9 01^{12} 01^2 0^2 1^9 01^{19} 01^2 01 01^6 01^{10} 01^4 01^{10} 01^{19} 01 0^2 1^6 01^4 01^{10}$
24	567 592	$01^{18} 01^8 0^2 1^{15} 01^8 01^{21} 0^2 1^8 01^{16} 01 0^2 1^6 01^2 01^{14} 01^{10} 01^{20} 01^{10} 01^{14} 01^2 01^6 0^2 101^{16} 01^8 0^2 1^{21} 01^8 0^2$ $1^{14} 0^2 1^2 01^3$ $1^{21} 01^{18} 01^{16} 01^4 01^7 01^6 01 01^{14} 01^3 0^2 1^8 01^7 01^3 01^2 01^{20} 0^2 1^3 01^7 01^{15} 01^7 01^3 0^2 1^{20} 01^2 01^3 01^7 01^9 01^{18}$ $01 01^6 01^{21} 01^7 01^5$
25	585 606	$1^{19} 01^9 01^{18} 0^2 1^{21} 01^{15} 0^2 101^{10} 01^{14} 01^{13} 01^5 01^9 01 0^2 1^8 01^6 01^8 0^2 101^9 01^5 01^{13} 01^{14} 01^{10} 01 0^2 1^{15} 01^{21}$ $0^2 1^4 01^9 01^2$ $1^{24} 01^4 01^{10} 01^{17} 01^8 01^4 01^{23} 01 01^{12} 01^5 01^{12} 01^9 0^2 1^3 01^8 01^6 01^{21} 01^6 01^8 01^3 0^2 1^9 01^{12} 01^5 01^{12} 01 01^{23}$ $01^4 01^7 0^2 1^3 01^{10} 01$
26	≥ 633 ≥ 642	$1^3 01^{13} 01^4 01^{12} 01^{20} 01^{21} 01^6 01^{18} 01^7 01^6 01^7 0^2 1^{12} 01^{14} 0^2 1^4 01^2 01^{14} 01^{25} 01^2 01^4 0^2 1^{14} 01^{12} 0^2$ $1^{14} 01^4 01^2 01^{13} 01^4 01^9 01^{18} 01$ $1^{15} 01^{23} 01^6 01^{13} 01^4 01^{14} 01^2 01^{13} 01^{11} 01^{15} 0^2 1^6 01^{12} 01^5 01^7 01^{14} 0^2 1^{11} 0^2 101^{13} 01^4 01^{17} 01^4 01^{13} 01^6$ $01^7 01^8 01^4 01^6 01^{13} 01^{17} 01^5$
27	≥ 663 ≥ 698	$1^{18} 01^{18} 01^{22} 01^5 01^{13} 0^2 1^{21} 01^4 01^9 0^2 1^2 01^8 01^2 01^3 01^8 0^2 1^{12} 01^{18} 01^9 01^{14} 01^5 01^9 01^{18} 01^{12} 0^2$ $1^{12} 01^{14} 0^2 1^9 01^2 01^{25} 01^4 01^4$ $1^{22} 01^6 01^{25} 01^{10} 01^{25} 01^{11} 0^2 1^9 01 01^5 01^3 01^7 0^2 1^9 01^{14} 01^{23} 01^3 0^2 1^{19} 01^{17} 0^2 1^3 01^5 01^{12} 01^{11} 01^7 0$ $1^9 0^2 1^7 01^{21} 01^{12} 01^{13} 01^6 01$
28	≥ 727 ≥ 742	$1^{26} 01^5 01^{13} 01^8 01^{19} 01^{13} 01^{10} 01^{19} 01^{12} 01^{15} 01^7 0^2 1^{12} 01 01^{12} 0^2 1^4 01^9 01^2 01^{15} 01^{22} 01^{15} 01^2 01^9 0$ $1^5 01^{12} 01 01^{13} 01^{14} 01^8 01^{12} 01^4 01^{10} 01^2$ $1^{21} 01^{22} 01 01^4 01^{25} 01^{10} 01 01^{23} 01^{11} 0^2 1^9 01^7 01^3 01^7 0^2 1^9 01^{14} 01^{27} 0^2 1^{19} 01^{17} 0^2 1^3 01^5 01^{12} 01^{11} 01^7$ $01^9 0^2 1^7 01^{21} 01^{12} 01^{13} 01^6 01$
29	≥ 809 ≥ 820	$1^{22} 01^{20} 01^{28} 01^{13} 01^3 01^6 01^4 01^{16} 01^{22} 01^{16} 01^{11} 01^{12} 0^2 1^2 01^7 01^5 01^{10} 01^{11} 01^3 01^5 01^{22} 01 01^3 01^{24} 0$ $1^{11} 01^3 01^6 01^9 01^{11} 01^{12} 01^{11} 01^2 01^{24} 0^2 1^3 01^{12}$ $1^{13} 01 0^2 1^{16} 01^{15} 01^{14} 01 01^4 01^2 01^{27} 01^4 0^2 1^{14} 01^{12} 01^{16} 01^6 01^7 01^{10} 01^7 01^6 01^{16} 01^{12} 01^{20} 01^9 01^4 01^{21} 0$
Continued on Next Page...		

Table 5: Good palindromic partitions for $\text{pdw}(2; 3, t)$

t	$\text{pdw}(2; 3, t) - 1$	Good palindromic partitions
		$1^{13}01^701^601^{13}01^60^21^{11}01^{29}01^{21}0101^701^601^4$
30	≥ 843	$1^{29}01^{25}0^21^{17}01^901^50^21^{22}01^701^{10}01^901^401^90^21^{17}01^{13}01^801^{28}01^501^{10}0101^{10}01^501^801^{19}0$ $1^{12}01^{24}01^{13}01^401^{10}01^201^{15}01^9010^21^{15}01^{11}$
	≥ 854	$1^{29}0101^20101^701^401^{16}01^{18}01^{14}01^{21}0^21^701^301^201^{13}01^{14}01^401^{16}01^301^{11}01^{24}01^401^{10}01^701^{25}01^{10}$ $0^21^401^{18}0^21^{15}01^{11}01^30101^401^301^{21}01^{18}01^901^{13}$
31	≥ 915	$1^{12}01^301^401^701^{11}01^801^{16}01^{10}01^401^301^401^{19}01^{29}01^601^{26}01^30101^{24}01^801^{16}010^21^{25}01^301^80$ $1^201^{16}01^20101^301^{24}01^701^301^{24}01^20101^{28}01^{20}01^{11}0101^{16}01^5$
	≥ 930	$1^{12}01^{19}01^{16}01^{10}01^901^301^401^{14}01^{12}01^{14}01^{18}0^21^{10}01^401^{13}01^801^50101^{10}01^{17}01^{15}01^{16}01^801^501^8$ $0101^401^801^{19}01^{16}01^501^{18}01^401^{25}01^{14}01^{16}01^301^901^{16}01^{13}01^{11}01$
32	≥ 957	$1^{24}01^301^701^{19}01^601^401^{17}01^{30}0^21^601^501^{24}0^21^{11}01^{18}01^401^{17}01^{19}01^{10}01^{11}01^701^{17}0^21^30$ $1^{23}01^{12}01^{11}01^70^21^301^{25}01^401^501^701^{22}01^701^{12}01^{30}01^201^601$
	≥ 962	$1^{11}0^21^{18}01^{21}01^501^201^{17}01^{19}0101^{16}01^{13}01^301^{24}01^501^201^{23}01^{12}01^{23}0^21^501^{24}0101^50^2101^901^{17}$ $01^{15}01^601^{19}01^901^{24}01^501^201^50^21^{23}01^{27}01^801^{19}01^2$
33	≥ 995	$1^{31}0101^{20}01^70^21^{12}01^401^{30}01^701^901^801^{11}01^301^{13}01^{18}01^{12}01^{11}01^30101^{10}0^21^401^701^{21}01^{14}01^{28}$ $0101^{11}01^30^21^{17}01^{30}01^60101^{28}01^{32}01^201^801^501^301^801^301^{13}$
	≥ 1004	$1^201^601^{22}01^401^{32}01^201^90^2101^701^501^{23}01^{18}01^{30}010^21^{16}01^{13}01^901^601^901^{19}01^{11}01^401^{17}01^{10}0$ $1^{26}0^21^{18}01^{10}01^{23}01^{12}01^{15}01^901^{12}01^60^21^80^21^301^{31}01^{12}$
34	≥ 1053	$1^901^{14}01^{21}0101^401^{27}01^801^{24}01^{32}01^301^401^{16}01^{13}0101^201^{33}01^301^401^301^{16}0101^{13}01^301^{16}01^6$ $01^{33}0^21^901^401^{31}01^20101^{16}01^401^{31}01^301^201^70^21^{22}0101^801^{30}01^3$
	≥ 1080	$1^{23}0^21^901^{22}01^{13}01^{25}0^21^2101^801^{15}01^{11}01^{18}0^21^401^{20}0101^201^501^{21}01^801^401^{23}01^401^801^{21}$ $01^{10}01^{20}01^40^21^201^{27}01^{10}01^401^201^{26}0101^{25}01^801^901^{17}01^40^21^{12}01^{21}0^21^301^3$
35	≥ 1113	$1^701^{12}01^{21}01^{28}01^301^{16}0101^{23}01^40101^{10}01^801^{14}01^{28}01^901^{12}01^{21}01^{23}0^2101^501^301^{21}01^201^{23}0$ $1^501^{14}01^{15}01^3010^2101^501^{17}01^{10}01^{23}0101^301^201^901^{21}01^{14}01^{30}01^{25}01^{10}01^7$
	≥ 1154	$1^{28}01^801^{18}01^601^901^{29}01^601^901^601^{12}01^50101^{10}01^{25}01^{28}01^{13}01^801^901^301^{25}01^{20}01^301^{21}01^50$ $1^{10}01^201^{34}0101^301^201^{15}01^{17}01^201^{27}01^{19}01^80^21^60101^20^21^{19}01^{30}01^{21}01^7$
36	≥ 1185	$1^601^{28}01^{10}01^{16}01^{28}01^{35}01^401^701^{23}0101^201^801^201^301^501^{15}01^20101^{19}01^{30}01^{25}01^{12}01^301^{14}0$ $1^801^{11}01^701^401^{18}0101^701^301^201^401^601^{21}01^301^{28}01^{24}01^{23}01^30^2101^{32}01^501^{11}01^{10}01^{15}$
	≥ 1212	$1^301^{25}01^{13}01^{24}01^501^401^{17}01^{13}01^{10}01^801^{28}010^21^{32}01^{10}01^{13}01^{20}0101^401^{12}0101^{13}01^{20}0101^801^401^{32}$ $0101^{28}01^401^{19}01^{32}01^601^{18}01^201^60^21^301^{24}01^601^{20}01^{13}01^{14}01^701^{23}01^401^8$
37	≥ 1271	$1^{30}01^601^{15}01^{13}01^{23}010^21^{15}0^21^{28}01^701^{28}01^201^301^{13}01^{22}0^21^{12}01^3010^21^{36}01^{28}01^{16}0^2101^{16}$ $01^{23}01^{13}01^801^301^901^{12}01^701^{29}01^{16}01^50101^{15}010^21^{33}01^{30}0^21^601^{12}01^{17}01^501^901^201^401^6$
	≥ 1294	$1^701^{24}01^{31}01^601^301^{30}0101^301^401^{22}01^{34}0101^801^{32}01^801^{22}01^201^701^201^301^{23}01^601^{22}01^{10}0101^4$ $01^901^{19}01^401^301^701^{15}01^{29}01^601^30101^201^{10}01^{22}01^{28}01^{12}01^801^{16}01^80101^801^{22}01^{23}01^401^{12}01^9$
38	≥ 1335	$1^{10}01^{35}01^{12}01^801^{12}01^70101^{33}01^{14}0101^501^{13}01^201^701^{22}01^{34}0^21^{13}01^701^{26}0101^401^{37}01^{13}0101^4$ $01^{36}0^2101^{27}01^{13}0101^701^{18}01^90^21^601^{29}01^{18}0^21^601^{28}0101^{21}0^21^{19}01^901^{36}01^601^6$
	≥ 1368	$1^{11}01^701^{36}01^{32}0^21^30^21^801^{20}0101^{11}01^{33}01^401^{16}01^801^{28}01^701^401^{26}01^{10}01^{22}01^401^801^401^{11}01^5$ $01^401^{11}01^{13}01^{24}01^{17}01^{13}01^401^{22}01^701^{17}01^801^90^21^401^{27}01^{20}01^{18}01^{22}01^301^90^21^{25}01^{10}01^{27}01$
39	≥ 1405	$101^{16}01^501^601^201^601^{33}01^401^{10}0^21^{28}01^601^401^{29}01^{18}01^601^{10}0101^{36}0^21^{22}01^{33}0101^{11}01^20101^{11}01^{22}$ $01^{33}01^401^{11}01^{21}0^2101^{13}01^301^{32}01^{19}01^{13}01^801^{11}01^{24}0^21^{15}01^{36}01^{13}01^601^{26}01^201^3010^21^{15}01^{11}01^4$
	≥ 1410	$1^301^{34}01^{24}01^301^{32}01^401^{18}0^21^{35}01^{12}01^701^{16}01^{11}01^601^901^{20}01^601^901^{19}01^501^{10}0101^{23}01^{16}0^21^{31}$ $01^301^{12}01^701^901^601^{28}01^601^{17}010^21^901^{18}01^60^21^801^201^{17}01^301^{32}01^501^{12}01^{34}0101^{24}01^{15}01^{13}01^301^7$

4.5. Open problems

- It seems the palindromic span can become arbitrarily large — also in relative terms? Perhaps the span shows a periodic behaviour, oscillating between small and large?

- Similar questions are to be asked for the gap. Does it attain value 0 infinitely often?
- Properties of the hypergraphs $\text{pdap}(t, n)$ seem to be of interest and non-trivial, starting with the estimation of the number of hyperedges and their sizes.
- Which structure of the palindromic problems makes them more amenable to conflict-driven SAT solvers than ordinary vdW-problems (see Subsection 5.1)?

4.6. Remarks on the use of symmetries

The heuristical use of symmetries for finding good partitions has been studied in [18, 20, 19]. Especially we find there an emphasise on “internal symmetries”, which are not found in the problem, but are imposed on the solutions.

The good palindromic partitions introduced in this section are more restricted in the sense, that they are based on the symmetries m from Subsection 4.1 of the clause-sets F expressing “ $w(k; t_0, t_1, \dots, t_{k-1}) > n$?” (i.e., we have $m(F) = F$; recall Subsection 1.1), which then is imposed as an internal symmetry on the potential solution by demanding that the solutions be self-symmetric. In [18] “reflection symmetric” certificates are mentioned, which for even n are the same as good palindromic partitions, however for odd n they ignore vertex 1, not the mid-point $\lceil \frac{n}{2} \rceil$ as we do. This definition in [18] serves to maintain monotonicity (i.e., a solution for $n + 1$ yields a solution for n , while we obtain one only for $n - 1$ (Lemma 4.1)). We believe that palindromicity is a more natural notion, but further studies are needed here to compare these two notions.

Further internal symmetries used in [18, 20, 19] are obtained by modular additions and multiplications (these are central to the approaches there), based on the method from [43] for constructing lower bounds for diagonal vdW-numbers. No generalisations are known for the mixed problems we are considering.

Finally we wish to emphasise that we do not consider palindromicity as a mere heuristics for finding lower bounds, but we get an interesting variation of the vdW-problem in its own right, which hopefully will help to develop a better understanding of the vdW-problem itself in the future.

5. Remarks on the use of SAT solvers

We conclude by summarising the experimental results and insights gained by running many SAT solvers on the many instances considered in this paper.

5.1. Complete solvers

Complete SAT solvers exist in mainly two forms, “look-ahead solvers” and “conflict-driven solvers”; see [42, 21] for general overviews on these solver paradigms. In general, for (ordinary) vdW-problems look-ahead solvers seem to perform better than conflict-driven solvers, while for palindromic problems it seems to be the opposite, however this might depend on the parameter setting, and with confidence we can only speak for the two problem classes considered in this paper. Considering publically available general-purpose SAT solvers:

- `satz` ([41], version 215), `OKsolver` (the version from 2002) and `march_pl` for look-ahead solvers,
- and `minisat` ([14]) versions 1, 2, 2.2.0, `picosat` version 913, `precosat` versions 236, 570, `glucose` version 1.0, and `cryptominisat`, version 2.9.0,

the overall “winner” on vdW-problems regarding $w(2; 3, t)$ amongst these solvers is `satz` (while from the conflict-driven solvers `minisat-2.2.0` and `glucose` perform best).⁴ We only considered $t \leq 16$, where on a current laptop for $t = 13$ `satz` takes about 10m, about 2h for $t = 14$, about 20h for $t = 15$, and about 200h for $t = 16$. If we assume a running-time factor of 10 for increasing t by 1, then for $t = 19$ it would take about 30 years (on a single medium-fast processor), which is better than the 200 years as reported in Section 2 (using a specialised, but more simple SAT solver), however this is likely misleading, due to the quadratic growth of the number of variables.

⁴We need to point out that this only holds for vdW-problems considering $w(2; 3, t)$. As already demonstrated in [36], with every parameter change a new game begins, and solvers behave very differently.

Showing $w(2; 3, 20) = 389$ (recall Subsection 2.2) seems to require new (algorithmic) insight. Since look-ahead solvers are good on this problem class, the theory of optimising heuristics as outlined in [34] is applicable, and should push the frontier.

The situation changes for palindromic vdW-problems, where conflict-driven solver perform much better than look-ahead solvers. `Minisat-2.2.0` and `precosat570` are clearly the strongest amongst the solvers we considered (the same as above), where perhaps for the largest problems `precosat570` gets the upper hand.⁵ For $t = 25$ and the hardest problem $n = 608$ it took about 10 days to determine unsatisfiability.

We remark that for $\text{pdw}(3, t)$ with odd n we can determine that the middle vertex $\frac{n+1}{2}$ can not be element of the first block of the partition (belonging to progression-size 3), since then no other vertex could be in the first block (due to the palindromic property and the symmetric position of the middle vertex), and then we would have an arithmetic progression of size t in the second block. Due to this (and there might be other reasons) palindromic problems for odd n are easier (running times can go up by a factor of 10 for even n).⁶

Using the new method developed in [37] (see also [22]), meanwhile we were able to verify the conjectured values for $\text{pdw}(2; 3, t)$ (recall Subsection 4.3) for $t = 26$ and $t = 27$ (still on a single computer!). In general it would be very interesting to find out why look-ahead solvers are good on ordinary vdW-problems and conflict-driven solvers on palindromic vdW-problems.

5.2. Incomplete solvers (stochastic local search)

In the `OKlibrary` we use the `UbcSAT` suite (see [49]) of local-search algorithms in version 1-2-0. The considered algorithms are `GSAT`, `GWSAT`, `GSAT-TABU`, `HSAT`, `HWSAT`, `WALKSAT`, `WALKSAT-TABU`, `WALKSAT-TABU-NoNull`, `Novelty`, `Novelty+`, `Novelty++`, `Novelty+p`, `Adaptive Novelty+`, `RNovelty`, `RNovelty+`, `SAPS`, `RSAPS`, `SAPS/NR`, `PAWS`, `DDFW`, `G2WSAT`, `Adaptive G2WSat`, `VW1`, `VW2`, `RoTS`, `IRoTS`, `SAMD`. The performance of local-search algorithms is very much instance-dependent, and so a good choice of algorithms is essential. Our experiments yield the following selection criteria:

- For standard problems (Section 2) the best advice seems to use `GSAT-TABU` for $t \leq 23$, to use `RoTS` for $t > 23$, and to use `Adaptive G2WSat` for $t > 33$ (also trying `DDFW` then).
- For the palindromic problems (Section 4) `GSAT-TABU` is the best algorithm.

Finally, using for vdW-problems a solution found for $n - 1$, and for palindromic vdW-problems a solution found for $n - 2$ (according to Lemma 4.1) as initial guess, helps to go much quicker through the easier part of the search space (of possible n), and also seems to help for the harder problems.

References

- [1] Dimitris Achlioptas. Random satisfiability. In Biere et al. [6], chapter 8, pages 245–270.
- [2] Tanbir Ahmed. Some new van der Waerden numbers and some van der Waerden-type numbers. *INTEGERS: Electronic Journal of Combinatorial Number Theory*, 9:65–76, 2009.
- [3] Tanbir Ahmed. Two new van der Waerden numbers: $w(2; 3, 17)$ and $w(2; 3, 18)$. *INTEGERS: Electronic Journal of Combinatorial Number Theory*, 10(A32):369–377, 2010.
- [4] Michael D. Beeler and Patrick E. O’Neil. Some new van der Waerden numbers. *Discrete Mathematics*, 28(2):135–146, 1979.
- [5] Armin Biere. Bounded model checking. In Biere et al. [6], chapter 14, pages 455–481.
- [6] Armin Biere, Marijn J.H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
- [7] T. C. Brown. Some new van der Waerden numbers (preliminary report). *Notices of the American Mathematical Society*, 21:432, 1974.
- [8] Tom Brown, Bruce M. Landman, and Aaron Robertson. Bounds on some van der Waerden numbers. *Journal of Combinatorial Theory, Series A*, 115:1304–1309, 2008.
- [9] V. Chvátal. Some unknown van der Waerden numbers. In R.K. Guy, editor, *Combinatorial Structures and their Applications*, pages 31–33. Gordon and Breach, New York, 1970.

⁵On smaller problems `glucose` is fastest, but from some medium size on `glucose` performs badly. For these instances, `precosat570` forgets learned clauses more aggressively than `minisat-2.2.0`, which seems to be advantageous here.

⁶We remark that while for example `precosat` determines this forced variables right at the beginning, this is not the case for the `minisat` versions, which infer that fact rather late, and they are helped by adding the corresponding unit-clause to the instance.

- [10] Scott Cotton. Two techniques for minimizing resolution proofs. In Strichman and Szeider [48], pages 306–312. ISBN-13 978-3-642-14185-0.
- [11] Adnan Darwiche and Knot Pipatsrisawat. Complete algorithms. In Biere et al. [6], chapter 3, pages 99–130.
- [12] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communication of the ACM*, 5:394–397, 1962.
- [13] Michael R. Dransfield, Lengning Liu, Victor W. Marek, and Mirosław Truszczyński. Satisfiability and computing van der Waerden numbers. *The Electronic Journal of Combinatorics*, 11(#R41), 2004.
- [14] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, Berlin, 2004. Springer. ISBN 3-540-20851-8.
- [15] Luís Gil, Paulo Flores, and Luís Miguel Silveira. PMSat: a parallel version of MiniSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:71–98, 2009.
- [16] Jun Gu. The multi-SAT algorithm. *Discrete Applied Mathematics*, 96-97:111–126, 1999.
- [17] Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Diversification and intensification in parallel SAT solving. In *CP’10 Proceedings of the 16th international conference on Principles and practice of constraint programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 252–265. Springer-Verlag, 2010.
- [18] P.R. Herwig, M.J.H. Heule, P.M. van Lambalgen, and H. van Maaren. A new method to construct lower bounds for van der Waerden numbers. *The Electronic Journal of Combinatorics*, 14(#R6), 2007.
- [19] Marijn Heule and Toby Walsh. Internal symmetry. In Pierre Flener and Justin Pearson, editors, *The 10th International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon’10)*, pages 19–33, 2010.
- [20] Marijn Heule and Toby Walsh. Symmetry within solutions. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 77–82, 2010.
- [21] Marijn J. H. Heule and Hans van Maaren. Look-ahead based SAT solvers. In Biere et al. [6], chapter 5, pages 155–184.
- [22] Marijn J.H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. Accepted by HVC 2011; available at <http://www.cs.swan.ac.uk/~csoliver/papers.html>, September 2011.
- [23] Marijn J.H. Heule and Hans van Maaren. Parallel SAT solving using bit-level operations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:99–116, 2008.
- [24] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.
- [25] Antti E. Hyvärinen, Tommi Junttila, and Ilkka Niemelä. Incorporating clause learning in grid-based randomized SAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:223–244, 2009.
- [26] Antti E. Hyvärinen, Tommi Junttila, and Ilkka Niemelä. Partitioning search spaces of a randomized search. In *AI*IA 2009: Proceedings of the XIth International Conference of the Italian Association for Artificial Intelligence Reggio Emilia on Emergent Perspectives in Artificial Intelligence*, volume 5883 of *Lecture Notes in Computer Science*, pages 243–252. Springer-Verlag, 2009.
- [27] Antti E. Hyvärinen, Tommi Junttila, and Ilkka Niemelä. Partitioning SAT instances for distributed solving. In *LPAR’10 Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, 2010.
- [28] Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of SAT solvers. *Journal of Automated Reasoning*, 34(1):73–101, January 2005.
- [29] Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Incomplete algorithms. In Biere et al. [6], chapter 6, pages 185–203.
- [30] Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In Biere et al. [6], chapter 11, pages 339–401.
- [31] Michal Kouril and Jerome L. Paul. The van der Waerden number $W(2, 6)$ is 1132. *Experimental Mathematics*, 17(1):53–61, 2008.
- [32] Daniel Kroening. Software verification. In Biere et al. [6], chapter 16, pages 505–532.
- [33] Oliver Kullmann. The OKlibrary: Introducing a “holistic” research platform for (generalised) SAT solving. *Studies in Logic*, 2(1):20–53, 2009.
- [34] Oliver Kullmann. Fundamentals of branching heuristics. In Biere et al. [6], chapter 7, pages 205–244.
- [35] Oliver Kullmann. Exact Ramsey theory: Green-Tao numbers and SAT. Technical Report arXiv:1004.0653v2 [cs.DM], arXiv, April 2010.
- [36] Oliver Kullmann. Green-Tao numbers and SAT. In Strichman and Szeider [48], pages 352–362. ISBN-13 978-3-642-14185-0.
- [37] Oliver Kullmann. Computing ordinary and palindromic van der Waerden numbers via collaboration between look-ahead and conflict-driven SAT solvers. In preparation, November 2011.
- [38] Oliver Kullmann. Constraint satisfaction problems in clausal form I: Autarkies and deficiency. *Fundamenta Informaticae*, 109(1):27–81, 2011.
- [39] Oliver Kullmann. Constraint satisfaction problems in clausal form II: Minimal unsatisfiability and conflict structure. *Fundamenta Informaticae*, 109(1):83–119, 2011.
- [40] Bruce Landman, Aaron Robertson, and Clay Culver. Some new exact van der Waerden numbers. *INTEGERS: Electronic Journal of Combinatorial Number Theory*, 5(2):1–11, 2005. #A10.
- [41] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI’97)*, pages 366–371. Morgan Kaufmann Publishers, 1997.
- [42] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [6], chapter 4, pages 131–153.
- [43] John R. Rabung. Some progression-free partitions constructed using Folkman’s method. *Canadian Mathematical Bulletin*, 22(1):87–91, 1979.
- [44] Stanisław P. Radziszowski. Small Ramsey numbers. *The Electronic Journal of Combinatorics*, August 2009. Dynamic Surveys DS1, Revision 12; see <http://www.combinatorics.org/Surveys>.
- [45] Vera Rosta. Ramsey theory applications. *The Electronic Journal of Combinatorics*, December 2004. Dynamic Surveys DS13, Revision 1; see <http://www.combinatorics.org/Surveys>.
- [46] Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Biere et al. [6], chapter 13, pages 425–454.

- [47] Tobias Schubert, Matthew Lewis, and Bernd Becker. PaMiraXT: Parallel SAT solving with threads and message passing. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:203–222, 2009.
- [48] Ofer Strichman and Stefan Szeider, editors. *Theory and Applications of Satisfiability Testing - SAT 2010*, volume LNCS 6175 of *Lecture Notes in Computer Science*. Springer, 2010. ISBN-13 978-3-642-14185-0.
- [49] Dave A.D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320, Berlin, 2005. Springer. ISBN 3-540-27829-X.
- [50] Hantao Zhang. Combinatorial designs by SAT solvers. In Biere et al. [6], chapter 17, pages 533–568.
- [51] Hantao Zhang, Maria Paola Bonacina, and Jie Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 11:1–18, 1996.

Appendix A. Using the OKlibrary

The OKlibrary, available at <http://www.ok-sat-library.org>, is an open-source research and development platform for SAT-solving and related areas (attacking hard problems); see [33] for some general information. For the purposes of this article the following components are directly relevant:

- The OKlibrary provides an already rather extensive library of functions for the computer algebra system Maxima. For example all hypergraph generators discussed in this article, and all vdW- and palindromic vdW-numbers can be computed and investigated at this level.
- For computations which take more time, C++ implementations are available.
- The OKlibrary provides easy access to a large number of (original) SAT solvers and related tools (as “external sources”).⁷
- Finally these components are integrated into tools for running and evaluating experiments.⁸

In the following sections we demonstrate the use of these tools. Some general technical remarks:

1. The installed OKlibrary lives inside directory OKplatform.
2. Anywhere inside this directory the Maxima-installation is called via `oklib --maxima` on the (Linux) command-line.
3. The C++ programs as well as the external sources, here the various SAT solvers, are placed on the path of the (Linux) user, and are thus callable by their name on the command-line (anywhere).

Appendix A.1. Numbers and certificates

All known vdW-numbers and palindromic vdW-numbers and known bounds are available at the computer-algebra level in the OKlibrary (using Maxima). For example the (known) numbers $w(2; 3, t)$ and $pdw(2; 3, t)$ are printed as follows (where inside the OKlibrary we typically use the letter “k” for the length of an arithmetical progression, not “t” as in this article):

```
OKplatform> oklib --maxima
(%i1) oklib_load_all();
(%i2) for k : 3 thru 40 do block([L:[3,k]],
    print(k,vanderwaerden(L), pdvanderwaerden(L), pd_span(L), pd_gap(L)));
3 9 [6,9] 3 0
4 18 [15,16] 1 2
5 22 [16,21] 5 1
6 32 [30,31] 1 1
7 46 [41,44] 3 2
8 58 [52,57] 5 1
```

⁷The aim is to serve as a comprehensive collection, also maintaining “historical” versions.

⁸In general we use the R system for statistical evaluation.

```

9 77 [62,77] 15 0
10 97 [93,94] 1 3
11 114 [110,113] 3 1
12 135 [126,135] 9 0
13 160 [142,155] 13 5
14 186 [174,183] 9 3
15 218 [200,205] 5 13
16 238 [232,237] 5 1
17 279 [256,279] 23 0
18 312 [299,312] 13 0
19 349 [338,347] 9 2
20 [389,inf-1] [380,389] 9 [0,inf-390]
21 [416,inf-1] [400,405] 5 [11,inf-406]
22 [464,inf-1] [444,463] 19 [1,inf-464]
23 [516,inf-1] [506,507] 1 [9,inf-508]
24 [593,inf-1] [568,593] 25 [0,inf-594]
25 [656,inf-1] [586,607] 21 [49,inf-608]
26 [727,inf-1] [634,643] 9 [84,inf-644]
27 [770,inf-1] [664,699] 35 [71,inf-700]
28 [827,inf-1] [[728,inf-1],[743,inf-1]] [15,0] [84,0]
29 [868,inf-1] [[810,inf-1],[821,inf-1]] [11,0] [47,0]
30 [903,inf-1] [[844,inf-1],[855,inf-1]] [11,0] [48,0]
31 [931,inf-1] [[916,inf-1],[931,inf-1]] [15,0] [0,0]
32 [1007,inf-1] [[958,inf-1],[963,inf-1]] [5,0] [44,0]
33 [1064,inf-1] [[996,inf-1],[1005,inf-1]] [9,0] [59,0]
34 [1144,inf-1] [[1054,inf-1],[1081,inf-1]] [27,0] [63,0]
35 [1205,inf-1] [[1114,inf-1],[1155,inf-1]] [41,0] [50,0]
36 [1258,inf-1] [[1186,inf-1],[1213,inf-1]] [27,0] [45,0]
37 [1339,inf-1] [[1272,inf-1],[1295,inf-1]] [23,0] [44,0]
38 [1379,inf-1] [[1336,inf-1],[1369,inf-1]] [33,0] [10,0]
39 [1419,inf-1] [[1406,inf-1],[1411,inf-1]] [5,0] [8,0]
40 unknown unknown unknown unknown

```

As one can see, if only bounds are known instead of a precise number x , then the number x is replaced by a pair (a, b) with $a \leq x \leq b$; “inf - 1” here indicates that the number is finite, but no more precise upper bounds are known.⁹ For palindromic span and gap the “0” indicates that these numbers could go up or down.¹⁰

Also the certificates (good partitions) are available, in various representations:

```

(%i4) cfull_certificate_string_pdvdw_3k(34);
(%o4) [[["1^{9}01^{14}01^{21}0101^{4}01^{27}01^{8}01^{24}01^{32}01^{3}0
1^{4}01^{16}01^{13}0101^{2}01^{33}01^{3}01^{4}01^{3}01^{16}0101^{13}
01^{3}01^{16}01^{6}01^{33}0^{2}1^{9}01^{4}01^{31}01^{2}0101^{16}01^{4}
01^{31}01^{3}01^{2}01^{7}0^{2}1^{22}0101^{8}01^{30}01^{3}"],
["1^{23}0^{2}1^{9}01^{22}01^{13}01^{25}0^{2}1^{21}01^{8}01^{15}0
1^{11}01^{18}0^{2}1^{4}01^{20}0101^{2}01^{5}01^{21}01^{8}01^{4}01^{23}
01^{4}01^{8}01^{21}01^{10}01^{20}01^{4}0^{2}1^{2}01^{27}01^{10}01^{4}0
1^{2}01^{26}0101^{25}01^{8}01^{9}01^{17}01^{4}0^{2}1^{12}01^{21}0^{2}
1^{3}01^{3}"]]]
(%i5) extract_data_certificates_pdvdw_3k(34);

```

⁹In principle there exist theoretical upper bounds, but for practical purposes these bounds are completely useless.

¹⁰At the time of printing this information we achieved to compute $\text{pdw}(2; 3, 26)$ and $\text{pdw}(2; 3, 27)$, using new methods; see the forthcoming [37].

```
(%o5) [[3,34],1054,1081,
      [[10,25,47,49,54,82,91,116,149,153,158,175,189,191,194,228,232,
        237,241,258,260,274,278,295,302,336,337,347,352,384,387,389,
        406,411,443,447,450,458,459,482,484,493,524]],
      [[24,25,35,58,72,98,99,121,130,146,158,177,178,183,204,206,209,
        215,237,246,251,275,280,289,311,322,343,348,349,352,380,391,
        396,399,426,428,454,463,473,491,496,497,510,532,533,537]]]
```

With the first command we get the representation of the good partitions as used in this paper (where now for the palindromic situation we have two good partition according to Corollary 4.1.3), while the second command yields a list with five elements: first the parameter tuple, then the two components of the palindromic number, and then two lists with the good partitions available, now represented via the block in the partition for the second colour.

Analysing the patterns according to Section 3, and applying these measurements to the certificates stored in the OKlibrary for $20 \leq t \leq 39$ is done as follows:

```
(%i6) for k : 20 thru 39 do
      print(k,firste(vanderwaarden3k(k)),
            map(analyse_certificate,full_certificate_vdw_3k(k)));
20 389 [[48,341],[44,45],[4,37],[5,27],[20,1]]]
21 416 [[50,366],[43,44],[7,34],[13,26],[8,1]]]
22 464 [[54,410],[51,52],[3,47],[5,40],[27,1]]]
23 516 [[59,457],[53,54],[6,46],[12,36],[17,1]]]
24 593 [[63,530],[57,58],[6,54],[13,37],[20,1]]]
25 656 [[74,582],[69,70],[5,64],[11,45],[16,2]]]
26 727 [[78,649],[72,72],[6,64],[13,42],[21,1]]]
27 770 [[79,691],[72,73],[7,65],[15,58],[11,2]]]
28 827 [[79,748],[74,75],[5,64],[11,44],[19,1]]]
29 868 [[81,787],[76,77],[5,69],[11,57],[27,1]]]
30 903 [[83,820],[76,77],[7,67],[13,57],[15,1]]]
31 931 [[82,849],[80,81],[2,77],[5,53],[58,1]]]
32 1007 [[87,920],[82,83],[5,78],[9,62],[29,1]]]
33 1064 [[89,975],[85,86],[4,80],[9,58],[25,1]]]
34 1144 [[96,1048],[87,88],[9,80],[19,63],[23,2]]]
35 1205 [[95,1110],[91,92],[4,84],[9,67],[41,1]]]
36 1258 [[101,1157],[97,98],[4,88],[9,72],[42,1]]]
37 1339 [[105,1234],[97,98],[8,90],[17,65],[30,2]]]
38 1379 [[104,1275],[96,97],[8,91],[17,73],[26,1]]]
39 1419 [[105,1314],[98,99],[7,95],[13,72],[46,1]]]
```

Per line we print out three items: t , the lower bound on $w(2; 3, t)$ and the list of data for each stored certificate. Now currently we have only stored one certificate for each $20 \leq t \leq 39$, and thus the third item contains just one list, with five pairs for the different statistics.¹¹ These five pairs have the following meaning:

1. First come n_0 and n_1 .
2. Then come the numbers of terms 0^s and 1^s (we don't use "00" here, and so these terms alternate, and thus their numbers differ at most by one).
3. Then from these counts the cases with $s = 1$ are excluded; thus the first element of the pair is n_{00} .
4. Now these exponents s are put in the list, and the sums of the numbers of peaks and valleys are computed; again for block 0 and block 1 of the partition, and thus now the second element of the pair is $n_p + n_v$.
5. Finally for these lists of exponents the maximal size of an interval with constant values is computed; thus if there were a second element of the pair with value 3 or greater, then Question 3.2 would have been answered in the positive.

¹¹We found more than one solution in each case, but always very similar to the one stored; there seems to be a clustering of solutions, and perhaps there is always only one (or very few) cluster.

Appendix A.2. Hypergraphs

The hypergraphs are available at Maxima-level, and the computational expensive palindromic hypergraph also at C++ level:

```
(%i8) arithprog_hg(3,5);
(%o8) [{1,2,3,4,5},{1,2,3},{1,3,5},{2,3,4},{3,4,5}]
(%i9) arithprog_pd_hg(3,5);
(%o9) [{1,2,3},{1,3},{2,3}]

> PdVanderWaerden-03-DNDEBUG 3 5
c Palindromised hypergraph with arithmetic-progression length 3
  and 5 vertices.
p cnf 3 2
1 3 0
2 3 0
```

Appendix A.3. SAT instances

The SAT-instance for considering $w(2; 3, t)$ with n vertices is created by the program call

VanderWaerdenCNF-03-DNDEBUG 3 t n,

for example for $t = 4$ and $n = 6$

```
> VanderWaerdenCNF-03-DNDEBUG 3 4 6
> cat VanDerWaerden_2-3-4_6.cnf
c Van der Waerden numbers with partitioning into 2 parts;
  SAT generator written by Oliver Kullmann, Swansea, May 2004, October 2010.
c Arithmetical progression sizes k1 = 3, k2 = 4.
c Number of elements n = 6.
c Iterating through the arithmetic progressions in colexicographical order.
p cnf 6 9
1 2 3 0
2 3 4 0
1 3 5 0
3 4 5 0
2 4 6 0
4 5 6 0
-1 -2 -3 -4 0
-2 -3 -4 -5 0
-3 -4 -5 -6 0
```

The SAT-instance for considering $pdw(2; 3, t)$ with n vertices is created by the program call

PdVanderWaerdenCNF-03-DNDEBUG 3 t n,

for example for $t = 4$ and $n = 9$

```
> PdVanderWaerdenCNF-03-DNDEBUG 3 4 9
> cat VanDerWaerden_pd_2-3-4_9.cnf
c Palindromic van der Waerden problem: 2 parts, arithmetic progressions of
  size 3 and 4, and 9 elements, yielding 5 variables.
p cnf 5 10
1 2 3 0
2 4 0
```



```

1 3 4 0
1 5 0
2 5 0
3 5 0
4 5 0
-2 -4 0
-1 -3 -5 0
-3 -4 -5 0

```

Appendix A.4. Running experiments

For running Ubsat-algorithm to determine lower bounds on $w(2;3,t)$ and $pdw(2;3,t)$, also providing “conjectures” on the precise values, we have the following tools (using no parameters here serves to print some basic helper-information):

```

> RunVdWk1k2
ERROR[RunVdWk1k2]: Six parameters k1, k2, n0, alg, runs, cutoff
are needed: The progression-lengths k1,k2, the starting number n0 of
vertices, the ubcsat-algorithm, the number of runs, and the cutoff.
An optional seventh parameter is a path for the file containing an
initial assignment for the first ubcsat-run.
> RunPdVdWk1k2
ERROR[RunPdVdWk1k2]: Five parameters k1, k2, alg, runs, cutoff
are needed: The progression-lengths k1,k2, the ubcsat-algorithm,
the number of runs, and the cutoff.

```

And for running complete solvers on palindromic instances we have

```

> CRunPdVdWk1k2
ERROR[CRunPdVdWk1k2]: Three parameters k1, k2, solver, are needed:
The progression-lengths k1, k2 and the SAT solver.

```